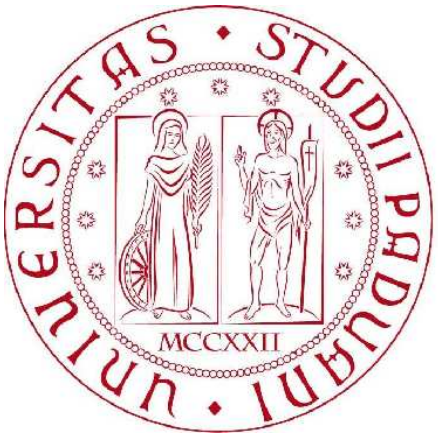
View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Padua@thesis

Università degli Studi di Padova



Facoltà di Ingegneria

Corso di laurea triennale in

Ingegneria Elettronica

TESI DI LAUREA

Modulatore di polarizzazione ottico tramite tecnologia LCD in trasmissione

STUDENTE Nicolò Zogno

RELATORE Prof. Gaudenzio Meneghesso

ADAPTICA

Progetto realizzato presso Adaptica S.r.l

INDICE

Capitolo 1
INTRODUZIONE

L'AZIENDA

Il lavoro è stato svolto all'interno dell'azienda Adaptica Srl nell'ambito dello stage formativo previsto nel piano di studi.
L'azienda nasce nel febbraio 2009 all'interno di M31, incubatore privato di aziende innovative operante nel settore delle tecnologie dell'informazione.
Adaptica raccoglie le scoperte e le tecnologie innovative sviluppate da un gruppo di ricercatori dell'Università di Padova nel campo dell'ottica e dell'optoelettronica.
Durante il percorso accademico di ricerca e progettazione, il team si è specializzato nell'integrazione di sistemi di ottica adattiva, in particolare nelle comunicazioni a singolo fotone ottimizzate con specchi deformabili, gli effetti non lineari e gli impulsi ultrabrevi.
Adaptica progetta e produce specchi deformabili e dispositivi opto-elettronici per l'ottimizzazione e il miglioramento di sistemi ottici.
I componenti ed i sistemi di ottica adattiva di Adaptica rappresentano una soluzione in tutti quei settori in cui l'immagine, ottenuta dai sistemi ottici, deve mantenere nitidezza, fuoco e fedeltà, anche nei dettagli. La tecnologia ed il sistema di lenti sviluppate da Adaptica riducono infatti le distorsioni e le aberrazioni, garantendo immagini più definite e nitide.
Questi componenti non solo potenziano le capacità del sistema ottico in cui vengono inseriti, ma risultano essere anche la soluzione di ottica adattiva che, per dimensioni e costi ridotti, meglio si integra in sistemi ottici in via di progettazione, aumentandone così le prestazioni.
Le tecnologie utilizzate, la metodologia di assemblaggio ed il costante controllo di produzione garantiscono qualità ed alte perfomance di tutti i prodotti.
Attraverso le più moderne tecnologie ottiche, elettroniche e meccaniche Adaptica realizza soluzioni opto-elettroniche all'avanguardia che trovano applicazione nel campo ottico, astronomico, dell'imaging, dell'automazione e delle misure e delle lavorazioni laser.

Alcuni dei prodotti di Adaptica

IDEA

Il sistema complessivo realizza un polarizzatore ottico: nel corso del tirocinio, tuttavia, coerentemente con le competenze del corso di studi, l'attenzione primaria è stata rivolta alla parte di *controllo elettronico*, che rappresenta il driver al quale viene connesso successivamente l'*elemento ottico*, ossia la *cella LCD* che fa da polarizzatore vero e proprio.

Una delle possibili applicazioni del prodotto finale potrebbe essere nell'ambito della fotografia, in quanto questo polarizzatore potrebbe essere facilmente integrato nelle fotocamere compatte e controllato attraverso il software interno, ad esempio tramite un menù a video.

Il progetto è stato sviluppato come prototipo, allo scopo di validare l'idea di base del polarizzatore elettronico: questo coincide con la prima fase di progetto del prodotto commerciale, che è solo uno studio di fattibilità, ma che non tiene conto di alcuni vincoli che dovranno invece essere rispettati nella versione definitiva.

In primo luogo non è stata rispettata alcuna particolare specifica riguardo alle dimensioni, essendo irrilevante qualora si stia analizzando solo la funzionalità complessiva del sistema.

Per questo motivo si è potuta utilizzare una scheda di sviluppo per FPGA Spartan 3A senza alcun problema.

PROGETTO

Per poter mettere in pratica questa idea, si è costruito un driver elettronico costituito dalle seguenti componenti:

- ✓ Scheda di sviluppo Xilinx Spartan 3A
- ✓ Circuito integrato Supertex HV892 alloggiato su un opportuno stampato

mentre il polarizzatore ottico in senso stretto è dato da una cella a cristalli liquidi connessa a tale elettronica di controllo.

Il chip Supertex HV892 costituisce già di per sé il driver della LCD, tuttavia l'FPGA è necessaria in quanto, una volta programmata, funziona da “ambiente di prova”.

Quindi, l'HV892 viene connesso alla Spartan, e da questa viene comandato mediante i vari switch a disposizione dell'utente.

In questo modo, la scheda di sviluppo Spartan *simula* un possibile ambiente di utilizzo, ad esempio una fotocamera, e funziona da interfaccia per l'utente.

OBIETTIVI CONSEGUITI

Nel corso del tirocinio, sono stati approfondite ed ampliate le conoscenze derivanti dal corso di laurea, riviste naturalmente sotto un profilo più pratico e commerciale; nel corso del progetto infatti si sono utilizzate componenti di vari produttori, pertanto si è dovuto effettuare uno studio delle relative specifiche tecniche allo scopo di garantirne la compatibilità con il resto del sistema.
Le competenze conseguite oppure approfondite nel corso del tirocinio sono sostanzialmente:

Ambito **PROGETTAZIONE ELETTRONICA**

- Linguaggio di descrizione dell'hardware **Verilog**

  al termine del tirocinio ho raggiunto una buona dimestichezza con l'utilizzo di tale linguaggio, prima completamente sconosciuto.

  Questo studio mi ha permesso di estendere le conoscenze già introdotte nel corso di Laboratorio di Elettronica Digitale, in cui si trattava il linguaggio VHDL
- Introduzione allo standard **Wishone**

  la conoscenza di questo standard per l'integrazione di blocchi logici è stata fondamentale per potermi servire dei progetti resi disponibili dalla community OpenCores, che si occupa di promuovere l'hardware open source
- Introduzione al bus **I^2C**

  il funzionamento di questo standard di comunicazione estremamente diffuso è stato studiato servendomi del documento ufficiale rilasciato dalla Philips Semiconductors, azienda ideatrice e promotrice di tale bus

Ambito **ECONOMICO – GESTIONALE**

- Analisi della concorrenza

  come attività extra rispetto al progetto del polarizzatore elettronico, e vista la mia curiosità anche verso la parte commerciale della produzione, l'azienda mi ha proposto la redazione di un documento, incluso nel business plan, con lo scopo di confrontare i maggiori concorrenti di Adaptica e fornire una veloce panoramica sulla tecnologia proposta dai diversi produttori di strumenti per l'ottica adattiva (specchi deformabili, correttori di fronte d'onda ecc..).

Capitolo 2
COMPONENTI

Introduzione

Il progetto completo è costituito fondamentalmente da una parte elettronica e da una parte ottica: la parte ottica è data dalla singola cella a cristalli liquidi, mentre la parte elettronica è un insieme di più componenti.
Per il pilotaggio della cella ci si è avvalsi di un circuito integrato prodotto dall'azienda americana Supertex, interfacciato mediante bus I^2C alla scheda di sviluppo Xilinx Spartan 3A.
Di quest'ultima, in particolare modo, sono stati sfruttati i dispositivi di input meccanici (switch, pulsanti, ruote..) e il display integrato, il tutto gestito dall'FPGA Spartan XC3S700A che, una volta programmato, costituisce il blocco logico del progetto.

2.1 - Scheda di sviluppo Xilinx Spartan 3A

Trattandosi di un prototipo per studiare la validità del progetto, si è scelta l'implementazione mediante una scheda di sviluppo basata su FPGA: infatti, l'intero progetto, una volta terminate tutte le procedure di test ed ottimizzazione, può essere completamente contenuto in un pcb di dimensioni molto ridotte, grazie all'utilizzo di un microcontrollore in luogo dell'FPGA.
Tuttavia, in questa prima fase di progetto, l'utilizzo di una evaluation board come quella della Xilinx è risultata molto più conveniente data la flessibilità offerta:
infatti, per prima cosa, l'operazione di programmazione della FPGA tramite USB è molto veloce agevolando quindi le operazioni di debug.
Come ulteriore vantaggio, tutto il restante hardware necessario, ad esempio gli switch e il display, è già presente direttamente sulla board.

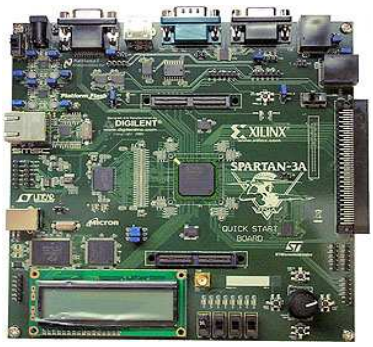


Fig. 2.1 – Xilinx Spartan 3A

Porte e componenti della Spartan 3A utilizzate

- ✓ FPGA XC3S700A-FG484
- ✓ Oscillatore al quarzo a 50 MHz
- ✓ Porta di download JTAG USB
- ✓ LED
- ✓ Switch a ruota
- ✓ Switch a scorrimento
- ✓ Switch a pulsante
- ✓ Display LCD a 16 caratteri x 2 linee

Di seguito vengono analizzate alcune di queste componenti, evidenziando le caratteristiche fondamentali, i principi di funzionamento, i loro limiti e gli accorgimenti che sono stati presi in fase di programmazione per consentirne un uso corretto.

2.1.1 – XC3S700A

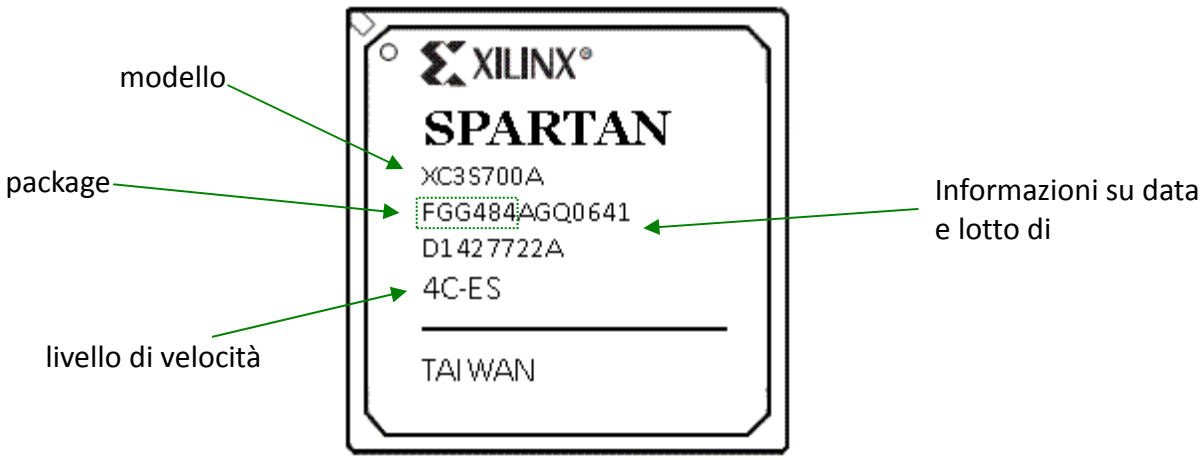


Fig. 2.2 – Xilinx Spartan 3A XC3S700A FPGA

2.1.2 – Oscillatore al quarzo

La board include un oscillatore al quarzo che opera alla frequenza di 50MHz, con un margine d'errore di circa 2500Hz.
L'uscita di questo componente è un'onda quadra di periodo 20ns, con la quale si sincronizzano la gran parte dei processi dell'FPGA.
La frequenza di questo oscillatore è sufficiente a soddisfare tutti i requisiti di velocità dei vari blocchi logici sincroni presenti nel progetto; anzi, in molti casi il problema è stato contrario: infatti si è dovuti ricorrere a dei divisori per ottenere un clock rallentato che si potesse adattare ai limiti di velocità imposti da certe componenti.

2.1.3 – Switch meccanici

Gli switch a scorrimento e i pulsanti sono idealmente molto semplici come funzionamento: infatti forniscono in uscita un segnale logico (1bit) che assume valore logico basso quando sono nella condizione di riposo, e diventa valore logico alto quando sono attivi.

STATO	VALORE LOGICO
Inattivo	Basso ('0')
Attivo	Alto ('1')

Il problema dei rimbalzi

Tuttavia, a causa della non idealità dei componenti, e dei tempi di transizione finiti rappresentati dalla pressione dei tasti da parte dell'utente, tale segnale non è pulito e non si comporta come un gradino.
Nel tempo in cui lo switch si trova in una posizione intermedia tra due livelli logici stabili, questo inizierà ad oscillare rendendo i fronti instabili: questa caratteristica può creare problemi qualora vi sia della logica sensibile ai fronti o ai livelli dello switch; ad esempio, se voglio che una determinata variabilie incrementi di uno ad ogni pressione di un pulsante, la presenza dei rimbalzi farà si che ad ogni singola pressione si verifichino molti più incrementi rispetto a quelli desiderati, rendendo impreciso ed imprevedibile il comportamento logico del circuito.

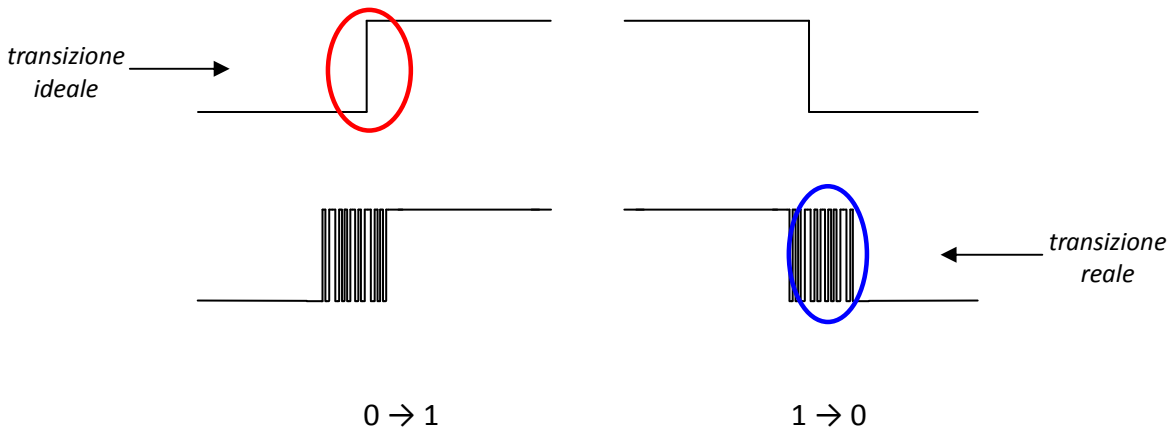


Fig. 2.3 – Transizioni logiche negli switch senza e con rimbalzo

Questo problema può essere ignorato nel caso in cui le aziende produttrici degli switch abbiano già previsto l'inserimento di un antirimbalzo hardware.
Tuttavia, i componenti presenti sulla evaluation board non sono dotati di questo sistema, pertanto è compito del progettista introdurre nella FPGA della logica aggiuntiva che corregga questo problema.
La soluzione adottata nel progetto viene presentata successivamente nella sezione dedicata ai moduli Verilog.

2.1.4 – Rotary knob (*Manopola*)

Un po' più complesso è il funzionamento della manopola a rotazione: questa è costituita da una una camma che va ad agire su due pulsanti distinti.
Il principio fondamentale è che questi pulsanti si attiveranno necessariamente in due momenti distinti, ed a seconda dell'ordine con cui vengono azionati si riesce a dedurre se la rotazione da parte dell'utente sia oraria o antioraria.
Dal punto di vista elettronico, questa periferica restituisce due segnali, detti 'A' e 'B', mappati su degli appositi pin della FPGA, che si comportano come nella figura seguente:

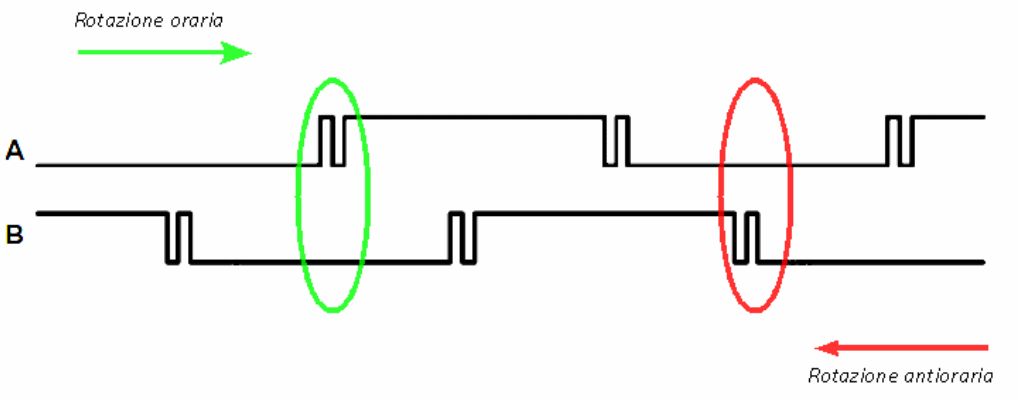


Fig. 2.4 – Andamento dei segnali A e B durante la rotazione

Ognuno di questi segnali indica lo stato di ciascuno dei due pulsanti, ad attivazione sfasata.
La figura evidenzia innanzitutto come sia presente anche qui il fenomeno dei rimbalzi.
Le due condizioni critiche sono evidenziate in verde e in rosso, e si individua una rotazione di tipo:

1. ORARIO
   quando il segnale 'A' compie una transizione '0' → '1' mentre il segnale 'B' è stabile a '0'
2. ANTIORARIO
   quando il segnale 'B' compie una transizione '0' → '1' mentre il segnale 'A' è stabile a '0'

2.1.5 – Sitronix display controller

La board include un semplice display a cristalli liquidi che può visualizzare fino ad un massimo di 16 caratteri per ciascuna delle 2 linee.
Si tratta di un prodotto economico ed a basse prestazioni, sia per la risoluzione che per il comportamento dinamico, che presenta una frequenza di refresh molto limitata se paragonata a quella dell'oscillatore al quarzo.
Tuttavia risulta perfetto per gli scopi del progetto, in quanto viene utilizzato soltanto per dare un breve riscontro scritto sullo stato del sistema.
L'interfaccia con il display è costituita da 4 segnali:

- LCD_E : è il segnale di enable, che abilita o disabilita il display
- LCD_DB : è il bus a 8 bit nel quale viaggiano i dati da o verso il display. Può essere anche utilizzato come bus a 4 bit semplicemente agganciando i 4 bit meno significativi a un valore logico alto.
- LCD_RS : indica se attraverso il bus a 8 bit stanno passando dati o istruzioni di registro
- LCD_RW : permette di selezionare se si intende effettuare una lettura o una scrittura

Prima di poter utilizzare il display è necessario eseguire una sequenza di operazioni con lo scopo di stabilire il protocollo di comunicazione.
Il processo richiede alcune decine di millisecondi e va eseguito una volta sola dopo l'accensione della evaluation board.

Sequenza di inizializzazione del display

1. Attendere 15 ms dopo la configurazione dell'FPGA.
   (circa 750.000 cicli di clock a 50Mhz)
2. Scrivere LCD_DB<7:4>=0x3, e LCD_E = '1' per 12 cicli di clock.
3. Attendere almeno 4.1 ms (circa 205.000 cicli di clock)
4. Scrivere LCD_DB<7:4>=0x3, e LCD_E = '1' per 12 cicli di clock.
5. Attendere almeno 100 µs (circa 5.000 cicli di clock)
6. Scrivere LCD_DB<7:4>=0x3, e LCD_E = '1' per 12 cicli di clock.
7. Attendere almeno 40 µs (circa 2.000 cicli di clock)
8. Scrivere LCD_DB<7:4>=0x3, e LCD_E = '1' per 12 cicli di clock.
9. Attendere almeno 40 µs (circa 2.000 cicli di clock)

All'interno del controller Sitronix sono presenti 3 regioni di memoria, alle quali si può accedere soltanto ad inizializzazione compiuta:

- DD RAM
- CG ROM
- CG RAM

DD RAM (Display Data Ram)

Questa memoria contiene 80 locazioni individuate ciascuna da un indirizzo esadecimale; ognuna di esse contiene un carattere del display, tuttavia soltanto 32 degli 80 totali vengono visualizzati.
Gli altri vengono tenuti in memoria e possono comparire, ad esempio, qualora si effettui uno scorrimento del testo; in particolare, la prima linea del display visualizza i caratteri contenuti nelle locazioni da 00 a 0F, mentre la seconda da 40 a 4F.

1	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	...	27
2	40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F	50	...	67
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	...	40

CG ROM (Character Generator Rom)

Si tratta di una memoria in sola lettura che contiene la mappatura di tutti i possibili caratteri visualizzabili a display.
Come mostrato in figura, quando si vuole visualizzare un determinato carattere nel display, non occorre far altro che eseguire l'algoritmo di scrittura caricando il codice binario a 8 bit del simbolo nella locazione di memoria DDRAM desiderata.

Ad esempio, se voglio visualizzare il simbolo "@" in basso a destra nel display, vado a scrivere l'indirizzo relativo a tale simbolo, cioè "01000000", nella locazione di memoria DDRAM "4F".

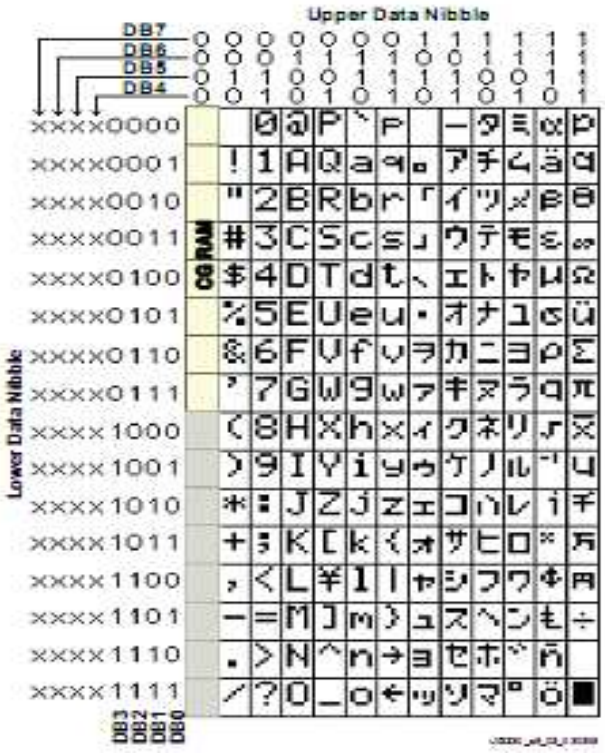


Fig. 2.5 – Mappatura dei simboli visualizzabili

CG RAM (Character Generator Ram)

Questa porzione di memoria non è stata utilizzata nel progetto.
Consente all'utente di memorizzare 8 simboli aggiuntivi (*custom*), identificandoli con una matrice di 8x5 che corrisponde graficamente alla matrice di pixel che costituiscono un carattere.
Per ogni posizione, viene scritto un '1' dove si vuole che il corrispondente pixel sia acceso, e uno '0' dove lo si vuole spento.

2.2 – Supertex HV892 (Liquid lens driver)

Per realizzare la parte relativa al pilotaggio della cella LCD, il progetto si avvale del chip HV892 prodotto dalla Supertex.
Questo dispositivo è stato originariamente concepito come driver per *lenti liquide*: il suo scopo sarebbe dunque quello di regolare la forma di una lente agendo sulla sua concavità (*menisco*).
Tuttavia è lecito utilizzare tale dispositivo anche per i nostri scopi: infatti, sia la lente liquida che la cella LCD si comportano come un carico capacitivo, al quale vogliamo applicare una differenza di potenziale tra le armature per modificare il campo elettrico.
L'unico accorgimento da tenere presente è quello che la capacità dell'LCD deve rientrare nel range fornito dalle specifiche, vale a dire al massimo 200pF.

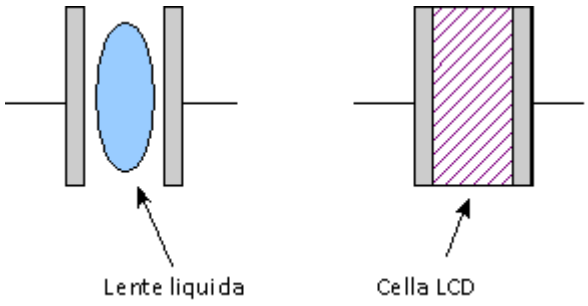
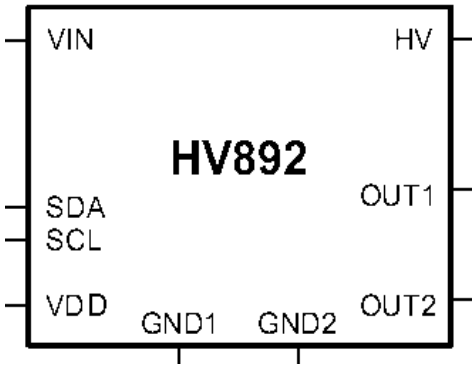


Fig. 2.6 – Equivalenza tra le strutture capacitive dei due dispositivi

Un primo approccio a scatola chiusa del dispositivo consente di capire meglio come sia stato integrato all'interno del sistema.
Di seguito vengono descritti i pin presenti sull'integrato:

V_{IN}	Tensione di ingresso
V_{DD}	Tensione esterna di riferimento per il bus I^2C
SCL	Bus I^2C : linea di clock
SDA	Bus I^2C : linea dati
GND1	Massa per il convertitore DC-DC
GND2	Massa per il resto del circuito
HV	Alta tensione in uscita dal boost integrato
OUT1	Uscite del driver: vanno connesse alle due armature della capacità di carico
OUT2	



Senza considerare alimentazioni e masse, e tenendo solo i segnali di dati il funzionamento del chip è il seguente: il dispositivo che fa da host pilota l'HV892 tramite l'interfaccia I^2C (ingressi SDA, SCL) trasmettendogli un codice univoco che corrisponde ad una tensione ben precisa ai capi della capacità di uscita, che è poi la cella LCD.
L'interfaccia I^2C del chip può funzionare a 3 livelli di velocità:

- **Normal**: 100Kbps
- **Fast**: 400Kbps
- **High speed**: 3.5Mbps

Vista la ridotta mole di dati che deve essere trasmessa nel presente progetto, si è deciso di predisporre l'integrato per il funzionamento a 100Kbps.

SCHEMA A BLOCCHI

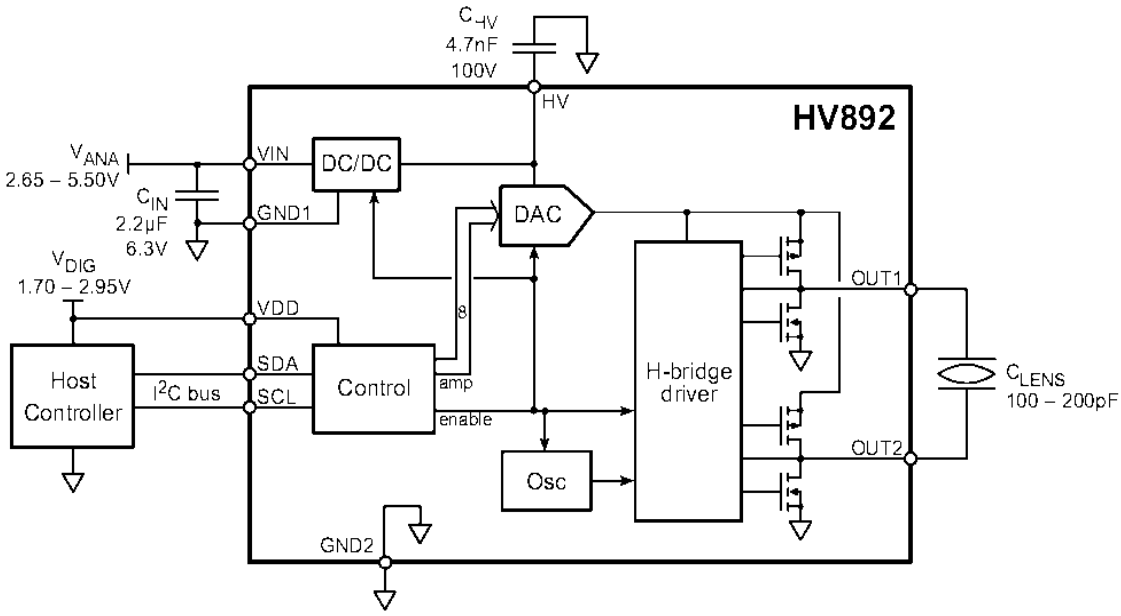


Fig.2.7 - Schema a blocchi del circuito integrato Supertex HV892.

Nello schema precedente il blocco *host controller* comprende il pcb su cui è alloggiato il chip e l'evaluation board stessa.
Il dispositivo *host* invia un comando, preventivamente codificato secondo il protocollo I^2C, tramite le due linee SDA e SCL, e viene ricevuto dal blocco *control*.
Questo si occupa di decodificarlo e trasformarlo da segnale seriale ad array di 8 bit: tale codice, che viene contrassegnato come *AMP,* è quello che mi consente di regolare l'ampiezza dell'onda quadra in uscita.
Poiché si tratta di un vettore ad 8 bit, ci permette di tarare l'uscita in 2^8 = 256 stati differenti secondo la seguente tabella:

Codice AMP *(esadecimale)*	**Descrizione**
00H	***Modalità STANDBY (risparmio energetico)*** L'intero dispositivo si porta ad uno stato di basso consumo, in cui il convertitore boost, il ponte H e l'oscillatore sono spenti, mentre i pin di uscita vengono connessi a massa.
01H ÷ FFH	Entro questo range di valori del vettore AMP l'ampiezza dell'uscita viene modulata. La ricezione di uno qualsiasi tra questi 255 valori comporta automaticamente l'interruzione dalla modalità di risparmio energetico.

La modulazione dell'ampiezza di uscita avviene in 255 differenti livelli con passo costante, secondo l'equazione

$$\mathbf{V_{OUT}} = 9.8\ \text{V} + AMP \cdot 205\ \text{mV} \qquad \text{(RMS)}$$

dove V_{OUT} è l'uscita del convertitore DAC, ottenuta combinando il segnale AMP con l'uscita del convertitore boost DC/DC.
Come si può notare, tale V_{OUT} va direttamente sul carico poiché rappresenta il valore logico alto del ponte H all'ultimo stadio, realizzato il logica CMOS statica.

PONTE H

Ridisegnando il ponte H presente nello schema a blocchi precedente in una forma più chiara ottengo il seguente circuito:

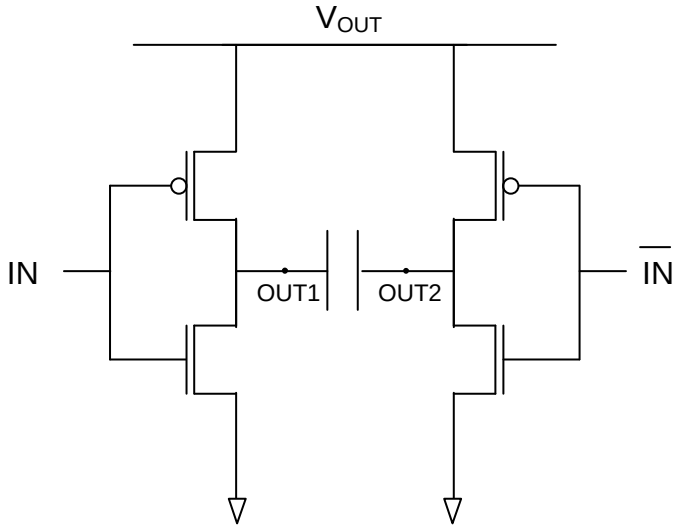


Fig. 2.8 – Ponte H realizzato in logica CMOS

Ciò non è altro che una coppia di inverter con gli ingressi complementari, le cui uscite sono collegate ai capi opposti della capacità di carico, che nello specifico del progetto è la cella LCD.
É fondamentale notare, prima di tutto, che il source dei pMOS è collegato alla V_{OUT}, cioè la tensione fornita dal convertitore DAC: questo significa che, quando V_{OUT} viene portata ad una delle due uscite OUT1 oppure OUT2 per l'accensione del relativo pMOS, in essa è inclusa automaticamente anche l'informazione che originariamente era stata ricevuta dal bus I^2C (cioè il valore AMP che rappresenta l'ampiezza dell'onda quadra generata).

Tale circuito può assumere solamente ***due stati*** (Fig. a pagina seguente).

Quando IN assume valore logico basso, il pMOS di sinistra si accende aprendo un cammino conduttivo tra OUT1 e V_{OUT} : in questo modo, dopo un tempo finito e non nullo richiesto per caricare/scaricare le capacità parassite, l'uscita si porterà al valore V_{OUT}.
Allo stesso modo, si accende anche l'nMOS di destra che, sempre dopo aver caricato o scaricato le capacità parassite, porta OUT2 a massa.
Quando al contrario, IN assume valore logico alto, si verifica la situazione complementare rappresentata in figura B.

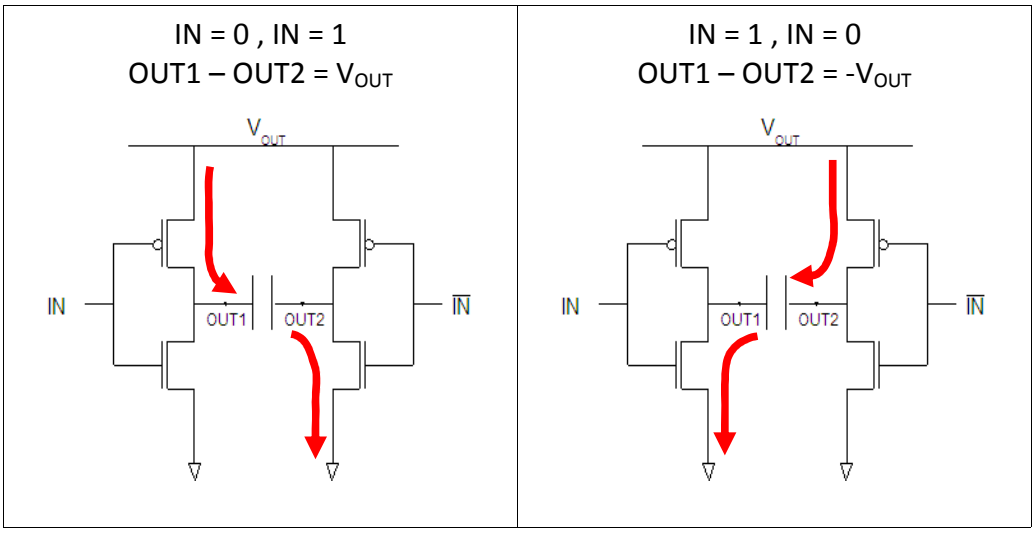


Fig. A *Fig. B*

L'***utilità*** di questo circuito si può intuire proprio da questo fatto: partendo da una tensione V_{OUT}, *positiva*, e avendo a disposizione l'ingresso IN, che può assumere due valori alto e basso ma comunque anch'essi *positivi*, si è in grado di applicare al carico una tensione con il segno voluto, cioè ± V_{OUT}.

Nel caso specifico del chip Supertex HV892, il ponte H è collegato ad un oscillatore con frequenza f_{OUT} nel range 1÷2 kHz, che genera un'onda quadra di ampiezza 2·V_{OUT} picco-picco.

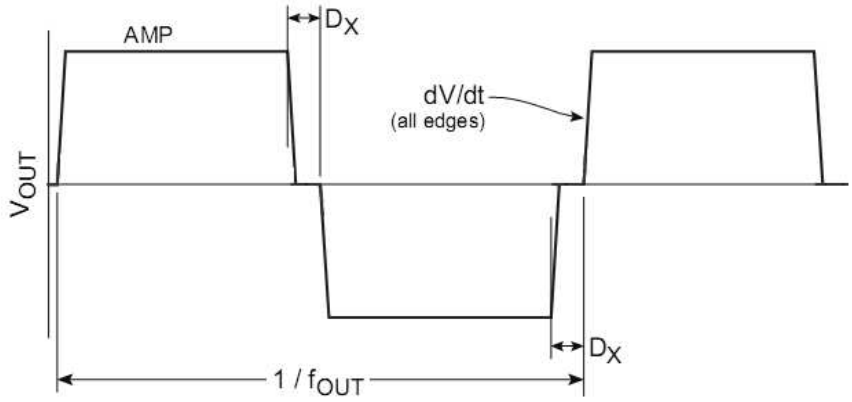


Fig.2.9 – Forma d'onda generata in uscita del chip, ai capi del carico.

Capitolo 3
PROGETTO VERILOG

Introduzione

In questa sezione verrà analizzata tutta la logica interna all'FPGA, che è stata descritta e sviluppata mediante linguaggio HDL.
La scelta del linguaggio è ricaduta sul Verilog, soprattutto per coerenza ed omogeneità con i precedenti progetti sviluppati da Adaptica.
Questa fase di progettazione è stata assistita dal software Xilinx ISE 11 Webpack, in dotazione con la evaluation board.
Si tratta di un ambiente di sviluppo integrato che funziona da: editor di testo, tool di simulazione, sintetizzatore di linguaggio ed interfaccia con la board.

3.1 – Mappa della struttura

La logica del progetto è suddivisa in diversi blocchi interconnessi secondo una gerarchia: questi blocchi sono detti *entities*, poichè ciascuno di essi realizza in maniera indipendente una precisa funzionalità.
Questo sistema si rivela molto produttivo ai fini della progettazione, semplificandola e rendendola più chiara, dal momento che consente di lavorare in maniera compartimentata, concentrandosi volta per volta nello sviluppo di una singola porzione della funzionalità complessiva.
Ciò che si ottiene è un progetto che include al suo interno diversi sotto-progetti fusi insieme tra loro.
Questa suddivisione è puramente concettuale, nel senso che al momento della sintesi il software della FPGA Xilinx provvede a tradurre queste strutture astratte in un insieme di interconnessioni e porte logiche elementari, perdendo le informazioni sulle varie entities che si erano create in fase di realizzazione.
La struttura tipica prevede la presenza di un modulo principale, detto *top;* questo viene posto al livello gerarchico più alto, caratteristica che gli conferisce due importanti funzioni:

1. i suoi ingressi e uscite sono quelli diretti all'esterno, pertanto possono essere *mappati* sui pin fisici dell'FPGA

2. il top è quel modulo che si occupa di richiamare e interconnettere le varie entities a livello gerarchico inferiore: tale operazione viene detta *istanza*

Nello specifico, il progetto di Nix è così strutturato:

core_logic_top.v **(TOP)**

- LCD_top.v **(DISPLAY)**
  - LCDEncoder.v
    - Bin2BCD.v
  - clock_slower2.v
  - LCDDisplay.v
    - LCDInterface.v
      - LCDcntrl.v
      - LCDDriver.v
- knob_interface.v **(INTERFACCIA)**
  - clock_slower.v
  - debounce.v
  - knob.v
  - ampgen.v
- controller_top.v **(WISHBONE MASTER)**
- i2c_master_top.v **(WISHBONE SLAVE)**
  - i2c_master_byte_ctrl.v
    - i2c_master_bit_ctrl.v

Nella seguente tabella viene riportato un rapido riassunto della funzione svolta dai blocchi più importanti:

ENTITIES PRINCIPALI	**DESCRIZIONE**
TOP (*core_logic_top.v*)	Si trova al livello gerarchico più elevato: la sua funzione principale è quella di connettere i vari blocchi inferiori e di interfacciarsi con i pin fisici. A tale scopo gli viene associato un file con lo stesso nome ma di estensione *".ucf"* , che permette di specificare a quali pin della FPGA debbano essere associati gli ingressi e le uscite del progetto. In questo caso il modulo top è praticamente vuoto, cioè non contiene alcuna logica aggiuntiva oltre alle consuete istanze dei moduli e a 2 buffer tristate che pilotano le linee SDA e SCL.
DISPLAY (*LCD_top.v*)	Questa entity si occupa della proiezione sul display delle informazioni sullo stato del sistema. È più articolata perchè contiene a sua volta diversi moduli: • un driver che provvede ad inizializzare e pilotare il display • un rallentatore di clock specifico per il display • un codificatore di stringhe. Il driver è il nucleo di questa entity, quello che (*continua..*)

	effettivamente comunica con il display; il rallentatore di clock è necessario poiché si tratta di un LCD economico a basse prestazioni, e non sarebbe in grado di funzionare ad una frequenza elevata come quella dell'oscillatore interno alla board, che lavora a 50MHz. Il codificatore di stringhe è la entity che si occupa di tradurre il dato che voglio visualizzare in una forma compatibile con l'input del driver: per svolgere questa operazione ci si serve anche di un convertitore binario → BCD, che consente di gestire agevolmente le singole cifre di un numero decimale.
INTERFACCIA (*knob_interface.v*)	Questa sezione del codice è adibita alla decodifica e alla corretta interpretazione dei segnali di input provenienti dai pin fisici dello switch a ruota. In aggiunta al modulo principale, che deve rilevare le rotazioni e stabilire se siano orarie, antiorarie, oppure se la ruota sia stata premuta al centro, sono presenti altre caratteristiche: prima di tutto è di fondamentale importanza il sistema antirimbalzo, che limita i danni provocati dalle transizioni non ideali dei segnali fisici provenienti dai sistemi di input meccanici. Oltre a questo, nella entity interfaccia è inglobato anche un blocco logico che genera il valore di AMP, inviato successivamente al dispositivo slave tramite I^2C.
WISHBONE MASTER (*controller_top.v*)	Il blocco Wishbone Master coordina il rispettivo Slave comunicandoli quali operazioni effettuare: nel caso specifico del nostro progetto, lo Slave deve solo effettuare scritture nell'I^2C bus ogni volta che si aggiorna il valore di AMP. Pertando questa entity è stata implementata con una macchina a stati finiti che esegue un ciclo di scrittura sullo slave ogni volta che AMP cambia.
WISHBONE SLAVE (*i2c_master_top.v*)	Questo modulo, pur essendo lo slave rispetto all'interfaccia Wishbone, si comporta da master rispetto al protocollo I^2C. È stato fornito pronto all'uso da OpenCores, community con lo scopo di condividere hardware open source, e promotore del bus Wishbone.

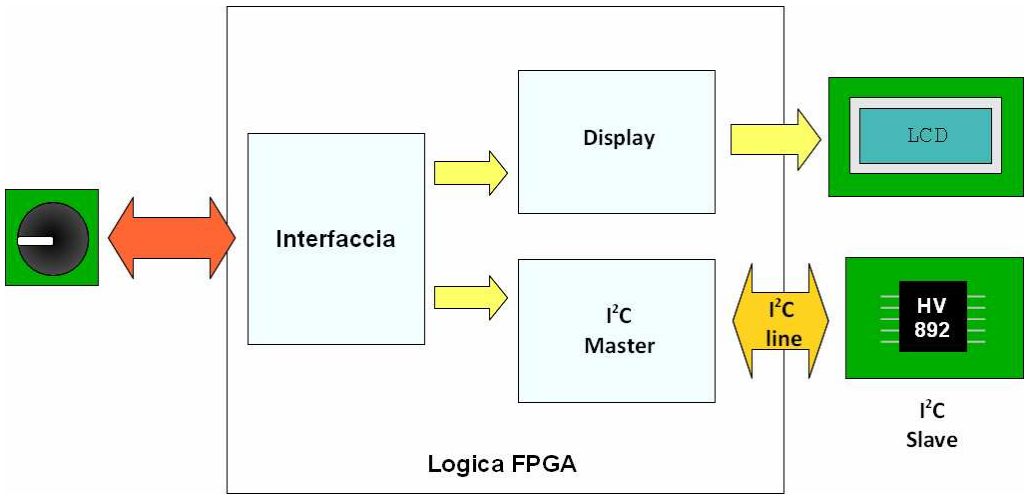
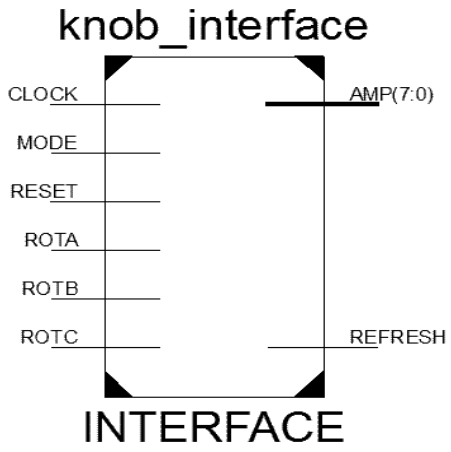


Fig. 3.1 – Schema riassuntivo delle varie entities

3.2 – Acquisizione degli input

Il blocco gerarchico più elevato di questa sezione è "*knob_interface.v*": analizzandolo per prima cosa a scatola chiusa è possibile farsi un'idea del suo compito e del comportamento che ha:

```
//Dichiarazione Verilog
module knob_interface(
    input ROTA,
    input ROTB,
    input ROTC,
    input MODE,
    input RESET,
    input CLOCK,
    output [7:0] AMP,
    output REFRESH
    );
```



Il modulo *knob_interface* riceve in ingresso 4 input provenienti da pin dell'FPGA associati a switch meccanici:

MODE = corrisponde allo switch a scorrimento *SW0*, che consente di selezionare tra la modalità di funzionamento continua oppure a 2 livelli, come verrà successivamente spiegato
ROTA, ROTB = sono i segnali provenienti dalla manopola girevole, e sono direttamente dipendenti dalla rotazione della stessa
ROTC = proviene anch'esso dalla ruota, e indica se questa è stata premuta al centro oppure no

mentre all'uscita ci sono soltanto 2 segnali:

AMP = è un bus ad 8 bit che può variare da 0 a 255, vale a dire gli stati che può assumere il polarizzatore
REFRESH = tale segnale vale normalmente '0', ad eccezione di quando viene azionato uno degli switch meccanici, poiché si genera un impulso (*strobe*) che viene poi propagato a tutti i blocchi logici successivi affinchè possano aggiornare il proprio stato
(*ad esempio, ogni volta che si rinnova il valore di AMP, si genera uno strobe che permette al display di aggiornare il valore visualizzato*)

Scendendo di un livello gerarchico si trovano tutte le altre sotto-entity che costituiscono questo blocco, e che combinati insieme ne vanno a costruire la funzionalità complessiva.
Si è utilizzata la entity di tipo ***debounce*** per realizzare il sistema antirimbalzo dei pulsanti: poiché il rimbalzo riguarda ogni switch meccanico, è stato necessario istanziare una copia del modulo per ogni dispositivo (4 copie).
Poi ancora si trova ***clock_slower***, che fornisce una versione rallentata del clock di sistema, utilizzata dall'antirimbalzo.

L'entity ***knob*** invece è quella che realizza la decodifica effettiva dei segnali verificando se l'utente stia compiendo una rotazione della manopola in senso orario oppure antiorario.
Il blocco ***ampgen*** svolge una funzione che non rientra propriamente nella categoria dell'interfaccia, tuttavia si è scelto di inserirlo qui per evitare di creare un ulteriore blocco a sé stante che avrebbe frammentato eccessivamente il progetto.

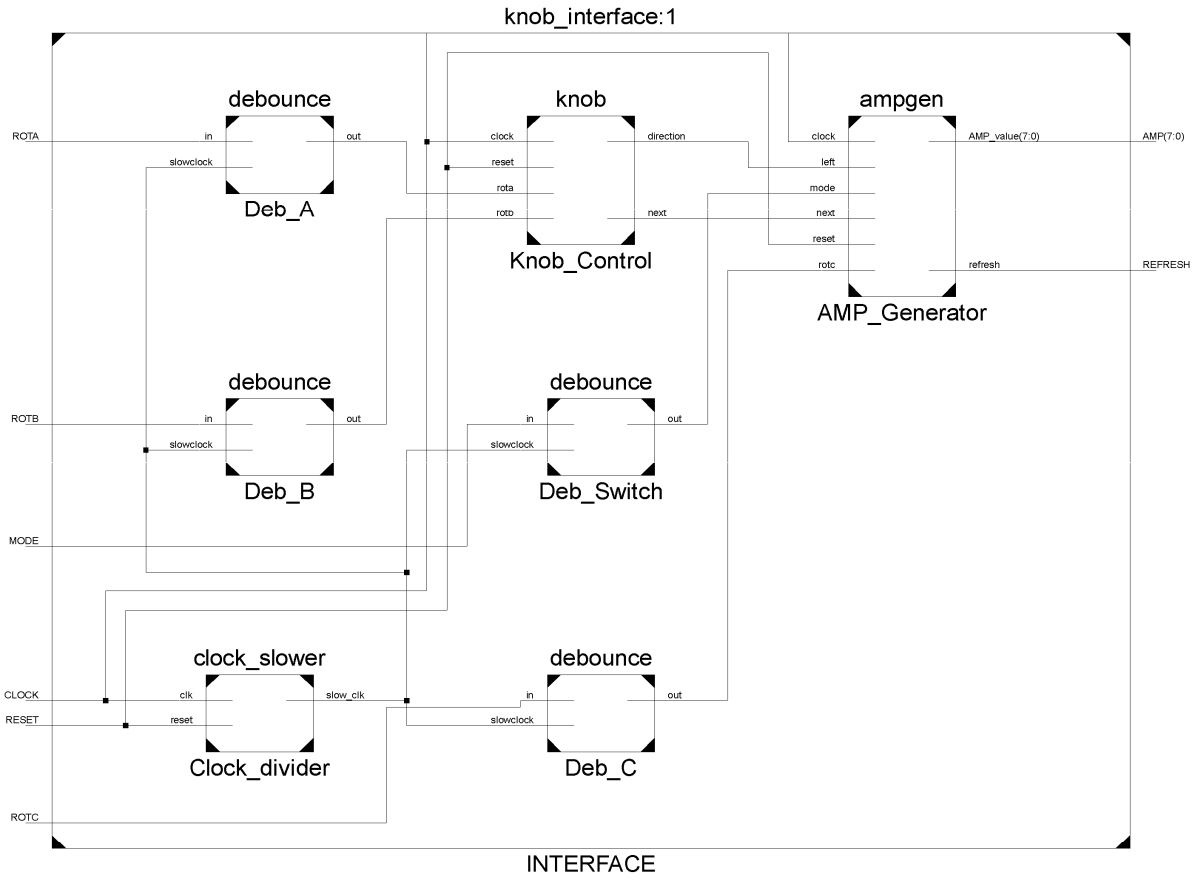


Fig. 3.2 – Rappresentazione "esplosa" della entity "knob_interface"

3.2.1 – Progettazione dell'antirimbalzo

La soluzione più semplice, veloce e funzionale per implementare un antirimbalzo è stata quella di sintetizzare un latch che campioni il segnale di ingresso (contaminato dai rimbalzi) all'uscita soltanto sul fronte di salita di un clock rallentato, restituendo così un segnale praticamente pulito.
Il codice Verilog che realizza questo blocco, estremamente compatto, è riportato di seguito:

```
module debounce(
    input in,
    input slowclock,
    output reg out
    );

always @ (posedge slowclock)
          out <= in;

endmodule
```

La generazione della versione di clock rallentata *slowclock* è avvenuta nel blocco *clock_slower* precedentemente citato, tramite il seguente codice:

```
module clock_slower(
    input clk,
    input reset,
    output slow_clk
    );

reg[15:0] contatore;

assign slow_clk = contatore[15];

always @ (posedge clk or posedge reset)
          if (reset)
               contatore <= 0;
          else
               contatore <= contatore + 1;

endmodule
```

Il principio di funzionamento è molto semplice: il vettore a 16 bit *contatore* viene incrementato di un unità ad ogni fronte positivo del clock di sistema.
L'uscita *slow_clock* non è altro che il bit più significativo di *contatore*, tant'è che viene effettuata un'assegnazione in continua.
Infatti, per un generico vettore *n*-dimensionale, che consente 2^n combinazioni, il bit più significativo vale '0' per i primi 2^{n-1} incrementi, mentre vale '1' per i successivi 2^{n-1}.
Il settaggio critico per il corretto funzionamento del sistema antirimbalzo (blocco *slowclock* + blocco *debounce*) è dato appunto dalla scelta del numero corretto di bit per il vettore *contatore*.
Dopo alcune prove pratiche, si è scelto come valore migliore 16 bit.
Questo si è rivelato il compromesso migliore tra ***affidabilità*** e ***prontezza***, dal momento che

- scegliendo valori *minori* di 16 il sistema presentava ancora problemi di rimbalzo
- scegliendo valori *maggiori*, il sistema non era in grado di catturare tutti gli input costringendo l'utente a ripremere più volte gli switch.

3.2.2 – Decodifica dei segnali fisici

In questo blocco logico vengono analizzati i segnali *rota* e *rotb* provenienti dalla manopola girevole, determinando se si tratta di una rotazione oraria oppure antioraria.
Questa informazione viene poi conservata e portata all'uscita tramite il registro *direction*, che vale '0' se si tratta di una rotazione oraria oppure '1' se la rotazione è antioraria.
A livello di linguaggio Verilog la soluzione è stata la seguente:

```
1     module knob(
2         input clock,
3         input reset,
4         input rota,
5         input rotb,
6         output reg next,
7         output reg direction
```

```
8         );
9
10    reg oldmerged;
11    assign merged = rota | rotb;
12
13    always @ (posedge clock)
14          if (reset)
15                begin
16                      oldmerged <= 0;
17                      next <= 0;
18                      direction <= 0;
19                end
20          else
21                begin
22
23                      oldmerged <= merged;
24
25                      if (oldmerged == 0 && merged == 1)
26                            begin
27                            next <= 1;
28                            if (rota == 1 && rotb == 0)
29                                  direction <= 0;
30                            if (rota == 0 && rotb == 1)
31                                  direction <= 1;
32                            end
33                      else
34                            next <= 0;
35
36                end
37
38    endmodule
```

Il primo passo è stato quello di assegnare a *merged* la funzione logica OR tra i due segnali *rota* e *rotb*.

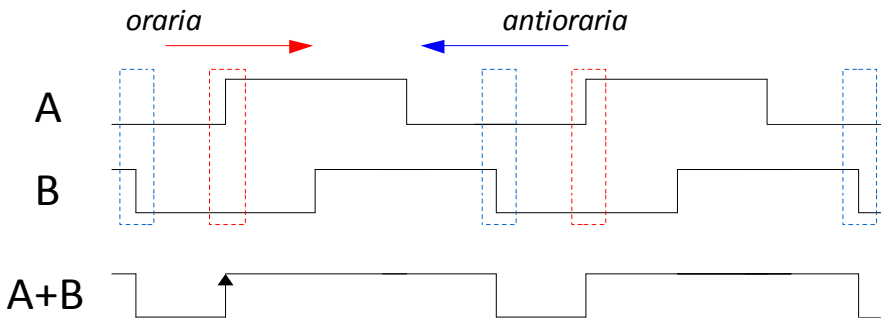


Fig. 3.3 – OR logico tra A e B

***NB:** il grafico va letto da sinistra a destra nel caso di rotazione oraria, mentre da destra a sinistra nel caso di rotazione antioraria: questo spiega perchè ciascuno scatto della ruota sia associato ad un fronte **positivo** del segnale A OR B.*

Come visibile nella figura 3.3, questo segnale ausiliario risulta essere molto utile, poiché ogni suo fronte di salita corrisponde ad uno scatto della manopola girevole (anche se non ci informa in alcun modo sulla direzione della rotazione!).
Sulla base di questa osservazione, il passo successivo è stato quello di inserire un costrutto logico che vada a verificare lo stato di *rota* e *rotb* ad ogni fronte positivo di *merged*.
Quest'operazione è situata tra le righe 28 e 31 del codice presentato, e la sua validità può essere facilmente verificata riferendosi nuovamente alla figura 3.3.

In sintesi, le uscite di questo blocco sono:

- **next**, un segnale impulsivo che si attiva ad ogni scatto della manopola
- **direction**, che vale '0' quando la rotazione è oraria e '1' quando antioraria

3.2.3 – Generazione del valore di AMP

Come già detto in precedenza, questo modulo non rientra propriamente nel blocco di acquisizione degli input: il suo compito è quello di mandare in uscita un valore binario a 8 bit, aggiornandolo in accordo con gli input ricevuti.
Di default, all'inizializzazione, il valore ***AMP*** è pari a 00xH, e corrisponde allo stato di basso consumo energetico dell'integrato Supertex HV892.
La entity ***ampgen*** è stata programmata in modo che il valore AMP di uscita risponda agli input con il seguente criterio:

- cresce ruotando la manopola in senso orario
- diminuisce ruotando in senso antiorario
- ritorna al valore di default premendo la ruota al centro
- il passo di incremento/decremento viene cambiato a seconda dello stato dello switch a scorrimento

La parte di codice è molto semplice da interpretare dal momento che fa uso esclusivamente di costrutti condizionali.

```
1  module ampgen(
2       input clock,
3          input next,
4       input reset,
5       input mode,
6       input left,
7       input rotc,
8          output reg [7:0] AMP_value,
9       output reg refresh
10     );
11
12 reg [7:0] temp_amp;
13 reg mode_copy;
14
15 always @ (posedge clock or posedge reset)
16     if (reset)
17           begin
18                 temp_amp[7:0] <= 8'h00;
19                 AMP_value <= 8'h00;
20                 refresh <= 0;
21           end
22     else
23           begin
24                 refresh <= 0;
25
26                 //Non appena attivo il modo 1 mi porto in standby
27                 mode_copy <= mode;
28                 if (mode_copy == 0 && mode == 1)
29                       begin
30                             temp_amp[7:0] <= 8'h00;
31                             AMP_value <= 8'h00;
32                             refresh <= 1;
33                       end
34                 if (mode_copy == 1 && mode == 0)
35                       begin
36                             refresh <= 1;
```

```
37                      end
38
39                //MODE 0 = modalità continua
40                if (next == 1 && mode == 0)
41                      begin
42                            refresh <= 1;
43                            if (left == 0 && temp_amp != 8'hFF)
44                                  begin
45                                        temp_amp <= temp_amp + 1;
46                                        AMP_value <= temp_amp + 1;
47                                  end
48                            if (left == 1 && temp_amp != 8'h00)
49                                  begin
50                                        temp_amp <= temp_amp - 1;
51                                        AMP_value <= temp_amp - 1;
52                                  end
53                      end
54                else
55                      begin
56                            if (rotc)
57                                  begin
58                                        refresh <= 1;
59                                        temp_amp[7:0] <= 8'h00;
60                                        AMP_value <= 8'h00;
61                                  end
62                      end
63
64                //MODE 1 = modalità 2 livelli
65                if (next == 1 && mode == 1)
66                      begin
67                            refresh <= 1;
68                            if (left == 0 && temp_amp == 8'h00)
69                                  begin
70                                  temp_amp <= `STEP1;
71                                  AMP_value <= `STEP1;
72                                  end
73                            if (left == 0 && temp_amp == `STEP1)
74                                  begin
75                                  temp_amp <= `STEP2;
76                                  AMP_value <= `STEP2;
77                                  end
78                            if (left == 1 && temp_amp == `STEP2)
79                                  begin
80                                  temp_amp <= `STEP1;
81                                  AMP_value <= `STEP1;
82                                  end
83                      end
84                else
85                      begin
86                            if (rotc)
87                                  begin
88                                        refresh <= 1;
89                                        temp_amp[7:0] <= 8'h00;
90                                        AMP_value <= 8'h00;
91                                  end
92                      end
93
94          end
95 endmodule
```

3.3 – *Approfondimento*: Wishbone Bus

Wishbone è un sistema di interconnessione tra più moduli funzionali (detti *core*) appartenenti a uno stesso circuito integrato: la sua ideazione risponde all'esigenza dei vari progettisti di poter condividere porzioni di descrizione hardware senza difficoltà di integrazione e compatibilità.
La sua applicazione trova particolare utilità nel campo dei dispositivi logici programmabili (PLD), dove l'hardware può essere condiviso sotto forma di linguaggio descrittivo dell'hardware (HDL), anziché sotto forma di schematici.
Infatti, mentre uno schematico descrive un dispositivo già a livello fisico e realizzativo, un codice HDL (*Verilog, VHDL..*) è solo una descrizione comportamentale della funzione che il circuito andrà a svolgere, e può essere reso specifico per una determinata architettura soltanto dopo un processo di sintesi: quindi il codice HDL gode di un'elevatissima *portabilità* e *riutilizzabilità*.
Il Wishbone nasce per rendere più semplice l'inserimento dei core di terza parte nei propri circuiti integrati, rendendosi particolarmente attraente in ambito open source.
Si dice pertanto che Wishbone è uno standard per l'integrazione System-On-Chip (SOC).
Il concetto del Wishbone non va confuso con quello dell'I^2C Bus: mentre l'I^2C è uno standard di comunicazione tra circuiti distinti, il Wishbone riguarda moduli funzionali nello stesso circuito.

PRECISAZIONE

- **Wishbone** = *bus interno ad uno stesso circuito*
  stabilisce delle convenzioni su nomi e funzioni dei segnali in uscita e in ingresso dai vari moduli funzionali in modo da poterli "agganciare" facilmente al resto del progetto

- **I^2C** = *bus di comunicazione tra integrati distinti*
  si tratta di un protocollo di comunicazione che impone un linguaggio preciso da rispettare per il trasferimento di dati tra più integrati: in questo modo si possono interfacciare componenti di produttori diversi

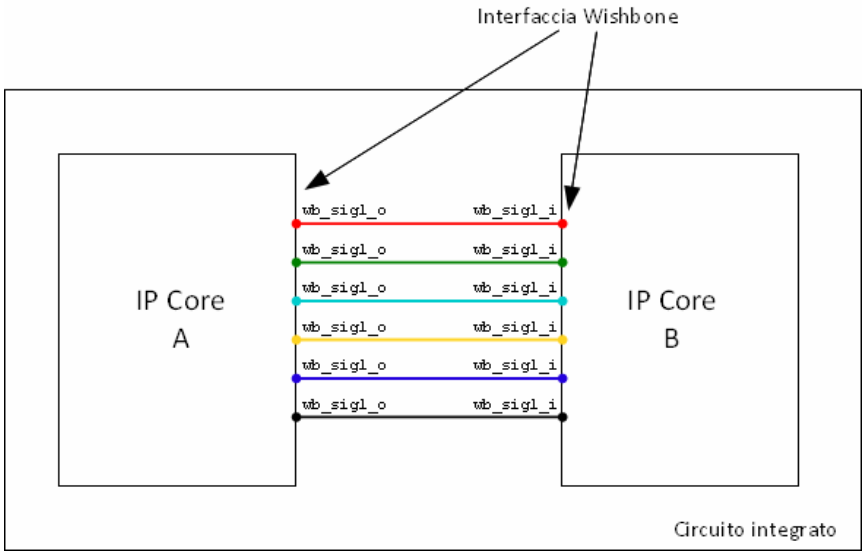


Fig. 3.4 – Interfaccia Wishbone tra due Core

I vantaggi più importanti derivanti dall'impiego dell'interfaccia Wishbone sono:

1. **Diffusione**
   l'utente può integrare direttamente blocchi logici sviluppati da altri progettisti
2. **Ottimizzazione**
   non c'è bisogno di logica collante (*glue logic*) per rendere compatibile l'hardware esterno
3. **Tempo**
   si possono riutilizzare moduli funzionali di terza parte con un approccio a "scatola chiusa", senza dover spendere tempo ad addentrarsi nei loro dettagli realizzativi

Nella pratica il Wishbone si traduce in un insieme di regole e convenzioni riguardanti il modo in cui un core si deve interfacciare con il resto della logica che lo circonda.
Questo comprende:

- **nomenclatura unificata** dei segnali esterni: tutte le connessioni del modulo devono attenersi a un preciso formato;
  ad esempio il nome `wb_rst_i` *sta ad indicare*
  1. che si tratta di un segnale compatibile con il Wishbone (prefisso `wb_`),
  2. che ha la funzione di reset sincrono(nome `rst`),
  3. che si tratta di un segnale in ingresso (suffisso `_i`)

- **documentazione**: in essa è viene descritto il modulo, con particolare attenzione sulla sua funzionalità, impiego e sulle dimensioni dei vari bus; questo per favorire la condivisione e il riutilizzo del core stesso, scopo primario del Wishbone

Nel caso specifico di questo progetto, si è potuto riscontrare in termini pratici l'utilità del protocollo Wishbone.
Infatti, per la gestione della linea I^2C, si è fatto uso di una entity di terza parte già sviluppata e testata.
Il fatto che aderisse alla convenzione Wishbone ("*Wishbone Compliant*") ha reso il processo di integrazione molto più agevole dato che, anche con una scarsa comprensione del suo funzionamento interno, è stato sufficiente considerare gli ingressi e le uscite verificandone il comportamento attraverso il documento delle specifiche Wishbone.

3.4 – *Approfondimento:* I²C Bus

3.4.1 - Introduzione

L' I²C è uno standard di trasmissione dati seriale tra circuiti integrati: questo permette di fare interagire tra loro dispositivi allocati nello stesso circuito stampato, anche se provenienti da diversi produttori, purché siano omologati a questo standard.
Intuitivamente, l'idea è quella che i vari produttori insegnino ad ogni dispositivo a parlare la stessa lingua, in modo che una volta collegati possano dialogare nativamente, senza nessun interprete.
I principali vantaggi di questa soluzione sono:

- *interconnessioni semplificate*
- *maggiore flessibilità*

INTERCONNESSION SEMPLIFICATE

L'I²C è un sistema di trasmissione seriale a 2 canali: un canale dati e un canale clock.
Questa caratteristica ha diversi vantaggi a livello di complessità strutturale: prima di tutto per quanto riguarda l'occupazione fisica, poiché il numero di pin nei circuiti integrati e di piste nei circuiti stampati risulta minimizzato.
Inoltre si guadagnano ulteriori spazio e risorse dall'eliminazione della cosiddetta *glue logic*, letteralmente *logica collante*, che si occupa di coordinare e gestire la comunicazione tra dispositivi con interfacce non unificate; ne sono un esempio tutti i decodificatori di indirizzo.

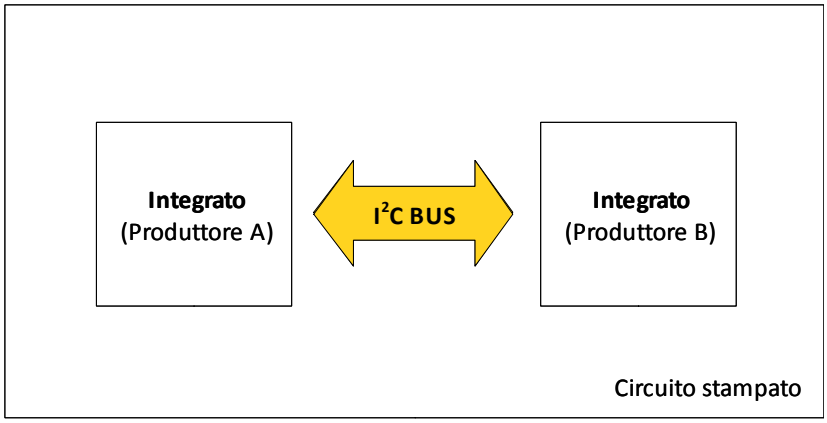


Fig. 3.5 – Universalità dell'I²C Bus

MAGGIORE FLESSIBILITÀ

L'adozione dello standard I²C nei propri sistemi garantisce oltretutto più semplicità per aggiornamenti e modifiche futuri: infatti se voglio sostituire un integrato più recente non sarà necessario riprogettare l'intero sistema.
Questo è possibile perché tutti i dispositivi I²C hanno sempre gli stessi due canali e pertanto se voglio aggiornare un determinato dispositivo non è necessario riprogettare tutto il sistema ma basta sostituirlo direttamente.
Supponiamo, ad esempio, di voler aggiornare un dispositivo integrato in un sistema che non adotti la convenzione I²C: nel caso in cui il numero o la funzionalità dei pin nel package sia variato non c'è più compatibilità con il resto del circuito, e va rivisto integralmente.
Si dice pertanto che un sistema basato su interfaccia I²C è *indipendente* dai dispositivi connessi.

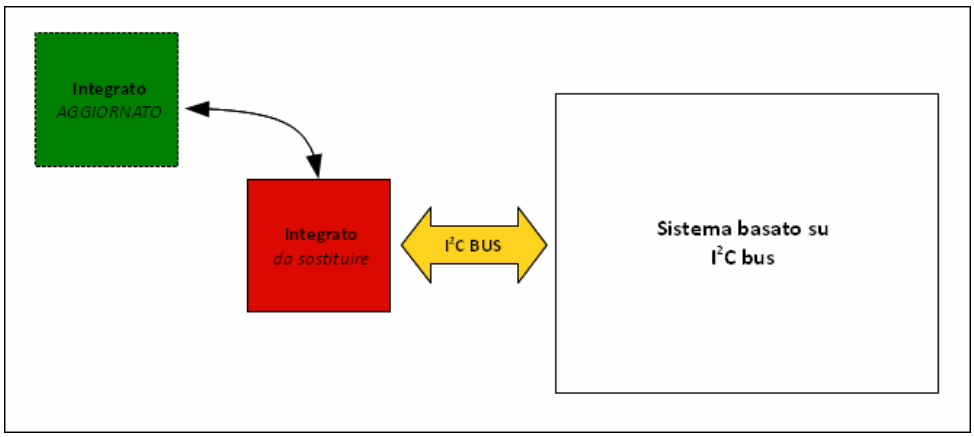


Fig. 3.6 – Indipendenza dai dispositivi connessi

3.4.2 - Funzionamento

Il bus I²C è costituito fisicamente da due linee, SDA e SCL: i dati effettivi passano attraverso la linea SDA (*serial data line*) in successione temporale scandita dai fronti del clock presente nella linea SCL (*serial clock line*).
Come regola generale la linea SDA viene considerata valida solamente durante il livello HIGH (valore logico 1) della linea SCL, e di conseguenza eventuali commutazioni del suo valore devono verificarsi solo durante lo stato LOW di SCL.

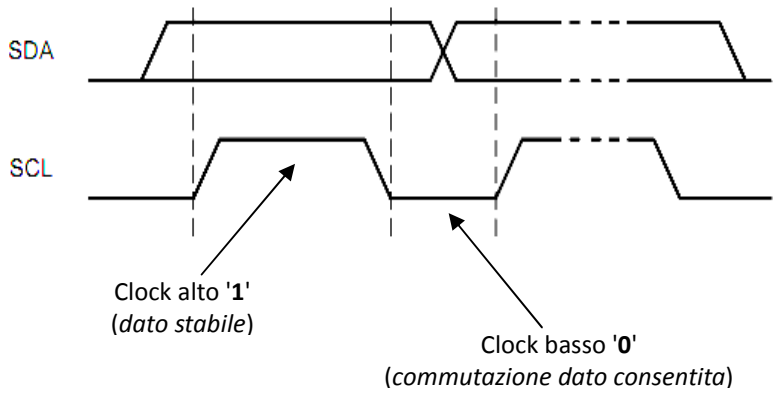


Fig. 3.7 – Comportamento della linea SDA in funzione della SCL

CONDIZIONI DI INIZIO E FINE TRASMISSIONE

Per delimitare una ben precisa sessione di trasmissione dati, il protocollo I²C prevede l'utilizzo di due condizioni univoche di inizio e fine (START e STOP) così definite:

- **START** se SDA ha una transizione 1 → 0 mentre SCL è stabile a 1
- **STOP** se SDA ha una transizione 0 → 1 mentre SCL è stabile a 1

Queste due condizioni rappresentano le uniche eccezioni consentite alla regola sulla validità dei segnali in quanto prevedono commutazioni di SDA quando SCL è alto.

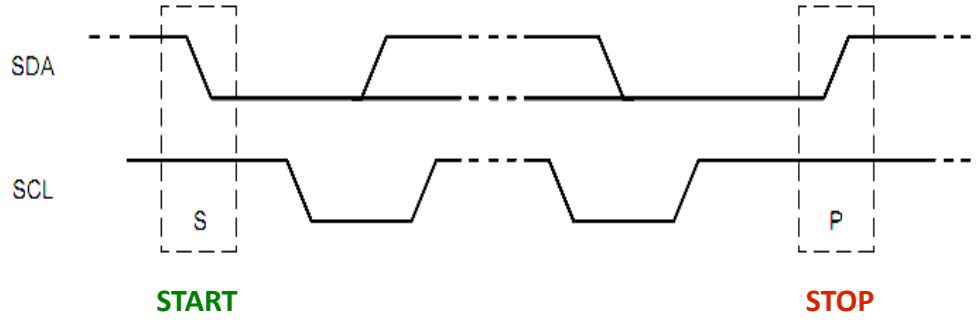


Fig. 3.8 – Condizioni di start e stop della trasmissione

NB In alternativa è possibile anche la condizione di *repeated START*, ossia uno start ripetuto, qualora il master voglia comunicare nuovamente con lo slave; a questo scopo è sufficiente che il master, in luogo dello STOP, invii un altro segnale di START.

TRASMISSIONE DEL DATO

Il protocollo I^2C trasmette pacchetti di dati a dimensione variabile, racchiusi entro le due condizioni di START e STOP; i pacchetti, seppur non abbiano limite di lunghezza, sono sempre costituiti dalla successione di uno o più byte.

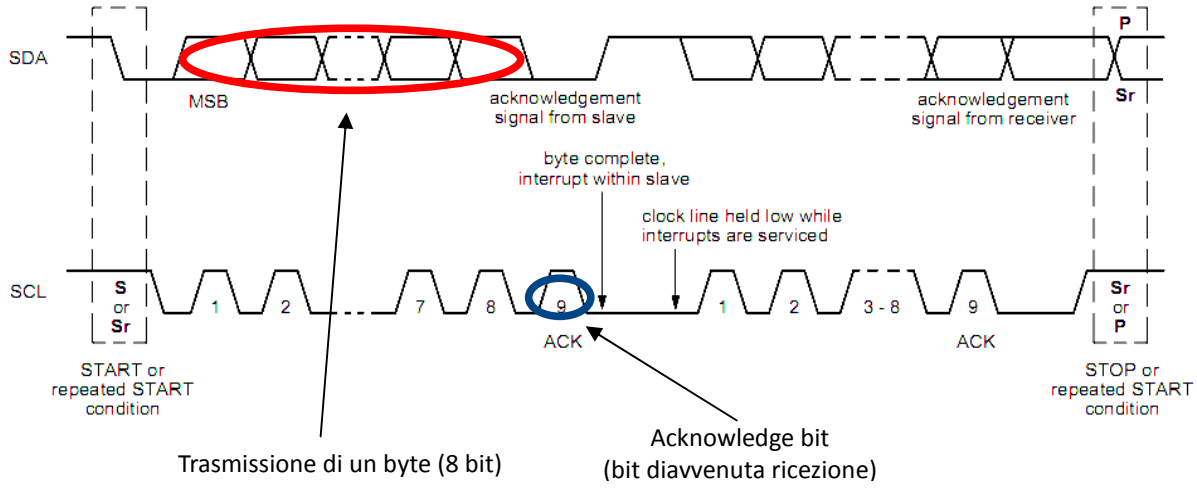


Fig. 3.9 – Trasmissione completa di un byte

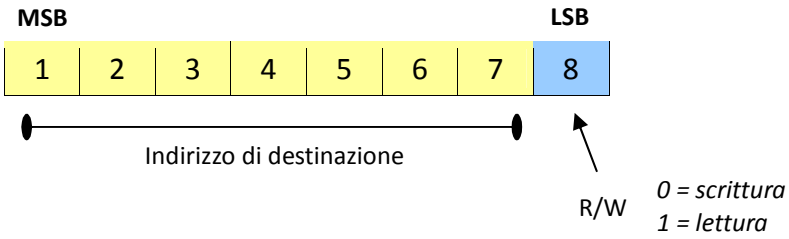
Per ogni byte che viene trasmesso, deve seguire uno speciale bit detto *acknowledge bit*, che ha lo scopo di confermare l'accettazione del dato da parte del dispositivo: la verifica ha successo se, mentre il dispositivo trasmittente (*master*) genera la 9° scansione di clock nella linea SCL, il dispositivo ricevente (*slave*) mantiene la linea dati SDA a livello logico basso.
Nel caso di una trasmissione non corretta, è compito del master decidere se annullare completamente l'invio o se ripetere l'operazione.

TRASFERIMENTI DI DATI

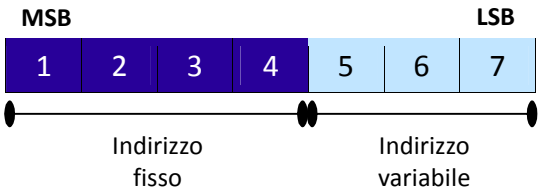
Il primo byte dopo la condizione di start ha generalmente la funzione di specificare quale dispositivo *slave* sarà coinvolto nella prossima trasmissione dati.
In esso sono infatti contenute informazioni riguardo a:

- L'*indirizzo* del dispositivo slave che si vuole contattare
- Il tipo di trasferimento da eseguire: *lettura* oppure *scrittura*

Questa indicazione avviene nel protocollo I^2C tramite l'invio di un byte speciale così strutturato:



Molto spesso capita di dover inserire nel medesimo sistema più dispositivi identici, tuttavia non si può accettare che questi presentino lo stesso indirizzo identificativo per evitare ovvi problemi di conflitto.
Questa esigenza può essere soddisfatta componendo gli indirizzi con una parte fissa, che indichi il tipo di dispositivo, e una parte variabile, per poter comunicare univocamente a quale delle diverse copie si sta facendo riferimento.
Ad esempio se considero il seguente indirizzamento a 7 bit secondo tale logica:



i quattro bit di indirizzo fisso permettono di individuare un determinato modello di dispositivo, mentre con gli altri 3 bit posso distinguere tra al massimo $2^3 = 8$ copie dello stesso prodotto inserite nel sistema.

3.5 – Controller I^2C

La parte più consistente di tutta l'architettura dell'FPGA è senza dubbio quella riguardante l'interfaccia I^2C.
Come già accennato, infatti, il chip Supertex HV892 che si vuole pilotare tramite la Spartan 3A è in grado di comunicare con l'esterno attraverso il bus I^2C: si è dovuto quindi implementare un modulo nella FPGA che fosse in grado di scrivere e leggere i dati trasmessi con tale bus.
La figura seguente consente di capire meglio l'architettura del controller I^2C realizzato tramite FPGA; all'interno sono individuabili due grandi moduli: *controller_top.v* ed *i2c_master_top.v*

- ***i2c_master_top*** = è il nucleo del controller complessivo, in quanto esegue tutte le comunicazioni secondo lo standard I^2C
- ***controller_top*** = funziona da master, secondo lo standard Wishbone, nei confronti della entity precedente: si occupa quindi di comandare quali comunicazioni debbano essere eseguite nel bus I^2C

OSSERVAZIONE 1
Dalla figura, si vuole rimarcare ancora una volta la differenza tra Wishbone e I^2C:
mentre il Wishbone serve a connettere due moduli interni all'FPGA, il bus I^2C si utilizza nelle connessioni fisiche verso l'esterno, quando vengono coinvolti due dispositivi distinti (in questo caso, la evaluation board da un lato, e l'integrato HV892 dall'altra).

OSSERVAZIONE 2
Il modulo *i2c_master_top* funziona contemporaneamente da **master** nei confronti del protocollo I^2C, mentre funziona da **slave** nei confronti del Wishbone.
Questo perchè dal fronte Wishbone deve ricevere le indicazioni su quali operazioni compiere, e conseguentemente sul fronte I^2C le deve eseguire.

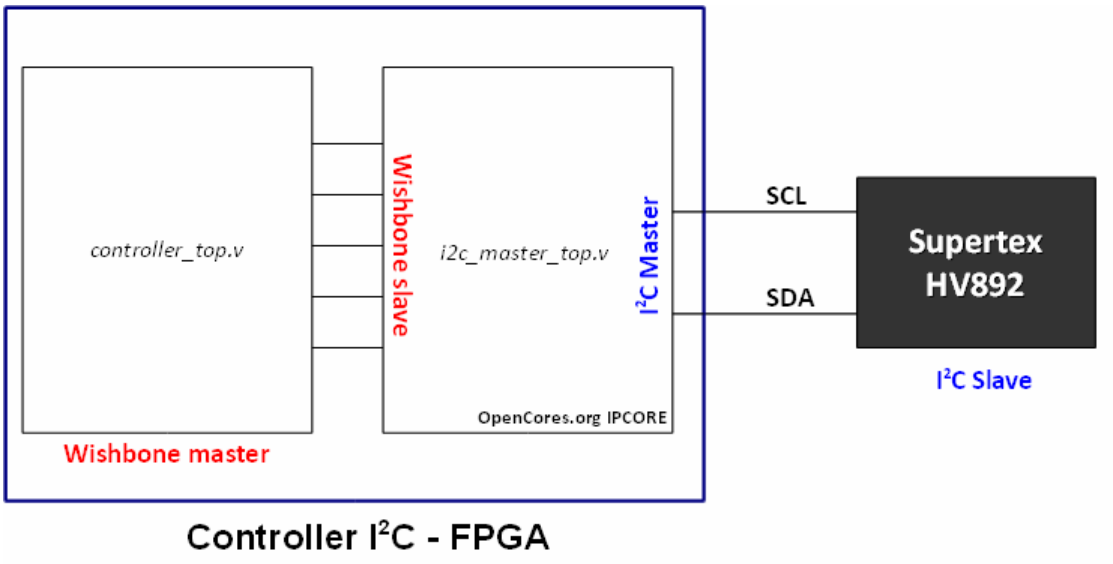


Fig. 3.10 – Schema riassunitvo del Controller I^2C implementato con FPGA

Scendendo nei dettagli realizzativi del controller, quasi tutta la logica è stata costruita basandosi sul sistema delle macchine a stati finiti: tale scelta è stata dettata principalmente dalla necessità di far ritornare all'occorrenza il sistema a una condizione precedente in maniera rapida.

Il funzionamento di questo controller non è infatti lineare, poichè è complicato notevolmente dal frequente inserimento di istruzioni condizionali che costringono il sistema a saltare da uno stato ad un altro: processi di questo tipo diventerebbero ingestibili se non si utilizzasse l'approccio della macchina a stati.

3.5.1 - OpenCores I²C Controller

Si tratta di un modulo generico per la gestione del protocollo I²C; nel progetto del polarizzatore, molte delle funzionalità di cui è dotato non sono state sfruttate: nel nostro caso, l'operazione che ci interessa eseguire nell'ambito del bus I²C è solamente la scrittura.
Infatti, richiamando quanto detto nei capitoli precedenti, l'FPGA non deve far altro che scrivere nel chip HV892 un valore *AMP* di dimensione 1 byte proporzionale all'ampiezza dell'onda quadra generata in uscita a tutto il sistema
Pertanto, almeno in questa prima fase di prototipo, non c'è la necessità di effettuare letture dallo slave I²C.
Il funzionamento del modulo I²C Master è basato su macchine a stati finiti, ed è implementato a più livelli di astrazione crescente.
Questo si può constatare notando che la entity *i2c_master_top.v* è dotata di altri due moduli annidati che svolgono funzioni sempre più a basso livello.

1. **LIVELLO WORD** (i2c_master_top) è in grado di gestire la comunicazione direttamente a livello di parole sfruttando le macchine a livello più basso
2. **LIVELLO BYTE** (i2c_master_byte_ctrl) ogni volta che viene interpellato compie la scrittura di un byte sullo slave I²C
3. **LIVELLO BIT** (i2c_master_bit_ctrl) è il livello gerarchico più basso, ed esegue singole operazioni di lettura/scrittura su bit singoli

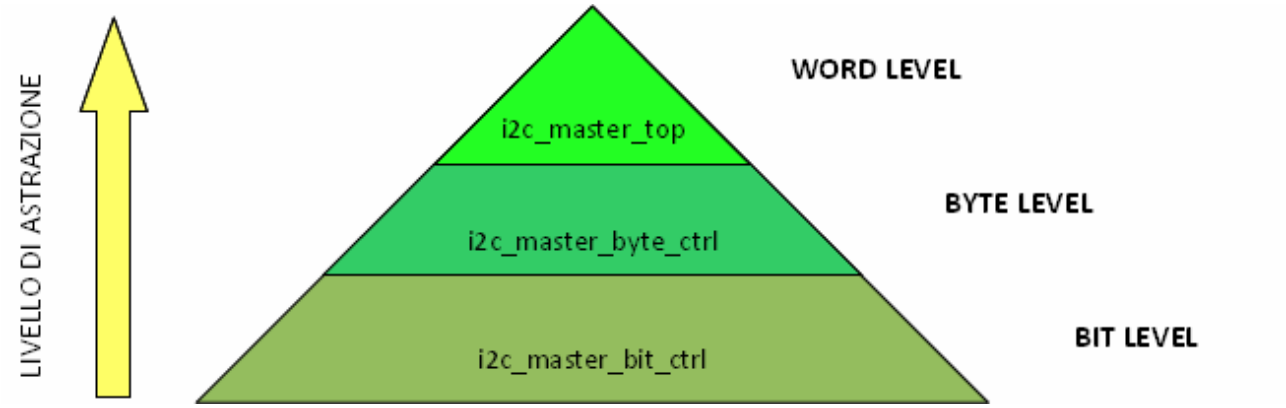


Fig. 3.11 – Livelli di astrazione delle entities I²C Master

Nel diagramma seguente (Fig. 3.12) è presentato uno schema a blocchi del funzionamento interno del blocco I²C Master, che include tutti i registri interni e i blocchi *i2c_master_byte_ctrl, i2c_master_bit_ctrl*.
Come si può osservare, la linea I²C viene modificata effettivamente soltanto dal blocco *i2c_master_bit_ctrl,* che essendo a livello più basso gerarchicamente, è quello progettato per compiere le operazioni di trasmissione più elementari.

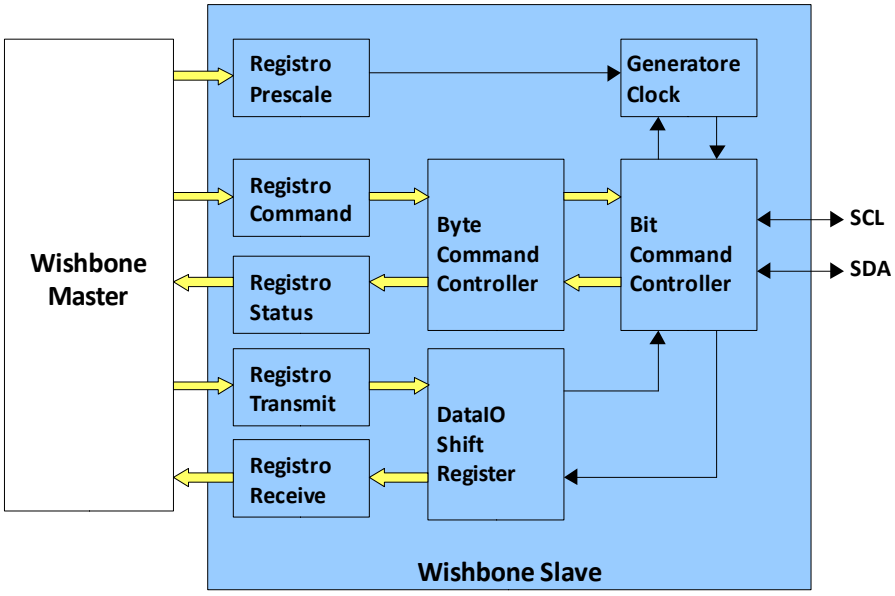


Fig. 3.12 – Schema a blocchi del funzionamento interno di I2C Master

Questo modulo mette a disposizione del master una serie di registri, che possono essere scritti o letti, attraverso il quale vengono realizzate tutte le operazioni.
Tali registri possono essere modificati o interrogati attraverso le seguenti porte del modulo:

- *wb_adr_i* (**Input**)
  Tramite questo bus a 3bit è possibile selezionare l'indirizzo del registro con il quale si intende interagire

- *wb_dat_i* (**Input**) & *wb_dat_o* (**Output**)
  Questi due bus a 8bit trasportano il byte coinvolto nell'operazione: in particolare, quando si intende scrivere in un registro, viene utilizzato il primo, mentre nel caso di lettura si impiega il secondo

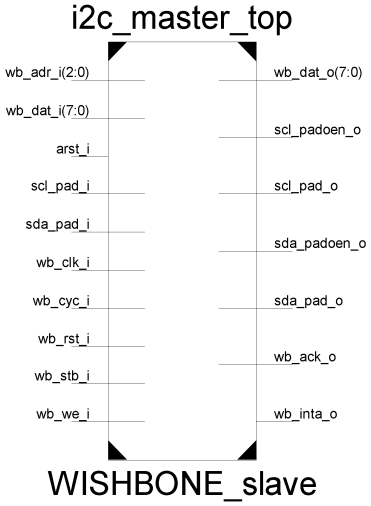


Fig. 3.13 – Schematico RTL

Viene presentata qui di seguito una panoramica di tutti i registri a disposizione, evidenziandone la funzione, la tipologia di operazioni che consentono e l'indirizzo che li individua:

Nome	Codice	Indirizzo	Dimensione	Accesso
Prescale	PRER_lo	0x00	8 bit	Read/Write
	PRER_hi	0x01	8 bit	
Control	CTR	0x02	8 bit	Read/Write
Transmit	TXR	0x03	8 bit	Write
Receive	RXR	0x03	8 bit	Read
Command	CR	0x04	8 bit	Write
Status	SR	0x04	8 bit	Read

PRESCALE REGISTER

Questo registro va solitamente settato all'inizializzazione del sistema, dal momento che il suo valore viene utilizzato come fattore di scala per dividere il clock dell'oscillatore e ridurlo a quello desiderato per la linea SCL (vale a dire il clock del bus I^2C).
Tale calcolo va eseguito in accordo con la formula fornita nelle specifiche del Controller I^2C:

$$prescale = \frac{1}{5}\frac{Clock\ sistema}{Frequenza\ SCL} - 1$$

Applicando tale formula al caso specifico del progetto, il clock di sistema è quello proveniente dall'oscillatore al quarzo, e vale 50Mhz, mentre la frequenza della linea SCL desiderata è di 100Kbps come precisato al capitolo 2.
Il fattore di prescale risulta pertanto pari a 99_{DEC} o in alternativa 63_{HEX}.
È proprio per consentire una maggiore regolabilità del clock I^2C che il valore di *prescale* viene memorizzato in due registri, anziché uno, così permettere numeri fino a 16 bit.

TRANSMIT REGISTER

È il registro utilizzato dal Wishbone Master per trasmettere il valore da scrivere nel Wishbone Slave. L'accesso è consentito solamente in scrittura, e può contenere 1 byte di dati.
Nel caso in cui si stia inizializzando la connessione con un dispositivo I^2C Slave, il byte a disposizione viene utilizzato per comunicare l'indirizzo del dispositivo fisico nei primi 7 bit più significativi, mentre il LSB viene sfruttato per indicare se si intenderà leggere ('1') o scrivere ('0') nello slave.

RECEIVE REGISTER

In questo registro, accessibile in sola lettura, viene presentato il byte proveniente dall'I^2C Slave in risposta ad una richiesta di lettura da parte del Wishbone Master.

COMMAND REGISTER

Si tratta di un registro accessibile in sola lettura, che il Wishbone Master utilizza per gestire la comunicazione. Il significato degli 8 bit che lo compongono è riassunto nel seguente prospetto:

Bit	Descrizione
7	Genera la condizione di start
6	Genera la condizione di stop
5	Lettura dallo slave
4	Scrittura allo slave
3	ACK, il WBM scrive '0' quando ha ricevuto un dato
2:1	*Utilizzo riservato*
0	IACK, il WBM scrive '1' per ripristinare dopo un'interruzione

STATUS REGISTER

Viene consultato in sola lettura dal Wishbone Master per aggiornarsi sullo stato della comunicazione nell'I^2C Bus.

Anche in questo caso, ogni bit ci dà informazioni su una precisa situazione, secondo il prospetto:

Bit	Descrizione
7	ACK, va a '0' quando la comunicazione con l'I^2C Slave ha avuto successo
6	BUSY, vale '1' nell'intervallo tra uno Start e uno Stop
5	*Non utilizzato (Arbitraggio)*
4:2	*Utilizzo riservato*
1	TIP, vale '1' durante un trasferimento, '0' quando è terminato
0	IF, interrupt flag, va a '1' quando il trasferimento non ha avuto successo

3.5.2 – Wishbone Master

Questo modulo funziona in modo complementare al Wishbone Slave appena presentato: mentre lo Slave non è altro che uno strumento in grado di svolgere tutte le operazioni possibili nell'ambito del bus I^2C, il Master è quel blocco che, in relazione alle esigenze del progetto, comanda e coordina tutti i compiti da svolgere: in termini specifici, lo Slave è in grado di realizzare le singole scritture e letture nel bus I^2C, mentre il Master opera a un livello più alto, scegliendo in quale ordine e in quale momento tali operazioni debbano essere eseguite.

Come già precisato più volte, il funzionamento del polarizzatore in versione prototipo prevede che l'unica operazione da effettuare nel bus I^2C sia quella di trasmettere il valore di *AMP* nel dispositivo I^2C Slave (Supertex HV892).

Quindi il Wishbone Master deve semplicemente scrivere il valore *AMP* nel Wishbone Slave provvedendo a trasmetterlo nuovamente ogni qualvolta il suo valore venga modificato dall'input dell'utente.

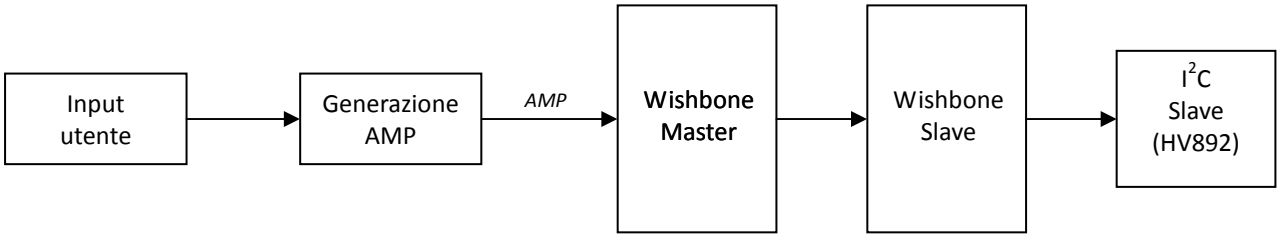


Fig. 3.14 – Schema a blocchi del sistema

Come evidenziato dalla fig. 3.14, nello schema a blocchi complessivo del sistema non sono previste frecce di ritorno (cioè letture) dal dispositivo I^2C Slave.

Il codice Verilog che realizza il modulo Wishbone Master è il seguente:

```
1 `include "timescale.v"
2 `include "define.v"
3 ////////////////////////////////////////////////////////////////////////////
4 //
5 // CONTROLLER
6 // Funziona da wishbone-master nei confronti del modulo i2c_master_top, che è
7 // il wishbone-slave
8 //
9 ////////////////////////////////////////////////////////////////////////////
10
11 module controller_top(
12 input trigger,
13 input [7:0] amp_value,
14 input wb_clk_i,
15 input wb_rst_i,
16 output reg [2:0] wb_adr_o,
17 output reg [7:0] wb_dat_o,
18 output reg wb_we_o,
19 output reg wb_stb_o,
20 output reg wb_cyc_o,
21 input wb_ack_i,
22 input wb_inta_i,
23 input [7:0] wb_dat_i
24 );
25
26
27 // Registri interni
28 reg [4:0] state; // FSM states
29 reg start_wb_cyc; // trigger for wishbone cycle
30 reg got_ack; // Set when ack has been received from wbm
31 reg [2:0] add_reg; // Register Address
32 reg [7:0] data_o; // Data register trasmission
33 reg [7:0] data_i; // Data register reception
34 reg w_rn; // Type of operation
35
36 //reg mem;
37 reg oldtrigger;
38
39 always @(posedge wb_clk_i or posedge wb_rst_i) begin
40 // MSF principale
41 if (wb_rst_i) begin
42 start_wb_cyc <= 1'b0;
43 state <= `INIT0;
44 add_reg <= 3'h0;
45 data_o <= 8'h00;
46 data_i <= 8'h00;
47 w_rn <= 1'b0;
48 end
49 else begin
50 case (state)
51 //INIZIALIZZAZIONE: registro di prescale
52 `INIT0: begin
53 add_reg <= `PRERlo;
54 data_o <= `PRESCALE;
55 w_rn <= 1'b1;
56 start_wb_cyc <= 1'b1;
57 state <= `INIT1;
58 end
59
60 `INIT1: begin
61 start_wb_cyc <= 1'b0;
62 if(got_ack)
63 state <= `INIT2;
64 else
65 state <= `INIT1;
66 end
67
68 `INIT2: begin
69 add_reg <= `PRERhi;
70 data_o <= 8'H00;
71 w_rn <= 1'b1;
72 start_wb_cyc <= 1'b1;
73 state <= `INIT3;
74 end
75
76 `INIT3: begin
77 start_wb_cyc <= 1'b0;
78 if(got_ack)
79 state <= `IDLE0;
```

```
80 else
81 state <= `INIT3;
82 end
83
84 //IDLE: la fsm resta in attesa di operazioni
85 `IDLE0: begin
86 oldtrigger <= trigger;
87 if (oldtrigger == 0 && trigger == 1)
88 state <= `ENABLE0;
89 else
90 state <= `IDLE0;
91 end
92
93 //ABILITO IL CORE
94 `ENABLE0: begin
95 add_reg <= `CTR;
96 data_o <= 8'b10000000;
97 w_rn <= 1'b1;
98 start_wb_cyc <= 1'b1;
99 state <= `ENABLE1;
100 end
101
102 `ENABLE1: begin
103 start_wb_cyc <= 1'b0;
104 if(got_ack)
105 state <= `WRITE0;
106 else
107 state <= `ENABLE1;
108 end
109
110 //SCRIVO L'INDIRIZZO DELLO SLAVE
111 `WRITE0: begin
112 add_reg <= `TXR;
113 data_o <= {7'b0100011 , 1'b0};
114 w_rn <= 1'b1;
115 start_wb_cyc <= 1'b1;
116 state <= `WRITE1;
117 end
118
119 `WRITE1: begin
120 start_wb_cyc <= 1'b0;
121 if(got_ack)
122 state <= `WRITE2;
123 else
124 state <= `WRITE1;
125 end
126
127 //SCRIVO NEL COMMAND REGISTER LA CONDIZIONE DI START
128 `WRITE2: begin
129 add_reg <= `CR;
130 data_o <= 8'b10010000;
131 w_rn <= 1'b1;
132 start_wb_cyc <= 1'b1;
133 state <= `WRITE3;
134 end
135
136 `WRITE3: begin
137 start_wb_cyc <= 1'b0;
138 if(got_ack)
139 state <= `WRITE4;
140 else
141 state <= `WRITE3;
142 end
143
144 //LEGGO LO STATUS REGISTER in attesa del TIP
145 `WRITE4: begin
146 add_reg <= `SR;
147 w_rn <= 1'b0;
148 start_wb_cyc <= 1'b1;
149 state <= `WRITE5;
150 end
151
152 `WRITE5: begin
153 start_wb_cyc <= 1'b0;
154 if(got_ack) begin
155 data_i <= wb_dat_i;
156 state <= `WRITE6;
157 end
158 else
159 state <= `WRITE5;
```

```
160 end
161
162 `WRITE6: begin
163 if (~data_i[1])
164 if (~data_i[7])
165 state <= `WRITE7;
166 else
167 state <= `STOP_ERR1;
168 else
169 state <= `WRITE4;
170 end
171
172 //SCRIVO IL DATO NEL TRANSMIT REGISTER
173 `WRITE7: begin
174 add_reg <= `TXR;
175 data_o <= amp_value;
176 w_rn <= 1'b1;
177 start_wb_cyc <= 1'b1;
178 state <= `WRITE8;
179 end
180
181 `WRITE8: begin
182 start_wb_cyc <= 1'b0;
183 if(got_ack)
184 state <= `WRITE9;
185 else
186 state <= `WRITE8;
187 end
188
189 //SCRIVO NEL COMMAND REGISTER LA CONDIZIONE DI STOP e WR BIT
190 `WRITE9: begin
191 add_reg <= `CR;
192 data_o <= 8'b01010000;
193 w_rn <= 1'b1;
194 start_wb_cyc <= 1'b1;
195 state <= `WRITE10;
196 end
197
198 `WRITE10: begin
199 start_wb_cyc <= 1'b0;
200 if(got_ack)
201 state <= `WRITE11;
202 else
203 state <= `WRITE10;
204 end
205
206 //LEGGO LO STATUS REGISTER bis
207 `WRITE11: begin
208 add_reg <= `SR;
209 w_rn <= 1'b0;
210 start_wb_cyc <= 1'b1;
211 state <= `WRITE12;
212 end
213
214 `WRITE12: begin
215 start_wb_cyc <= 1'b0;
216 if(got_ack) begin
217 data_i <= wb_dat_i;
218 state <= `WRITE13;
219 end
220 else
221 state <= `WRITE12;
222 end
223
224 `WRITE13: begin
225 if (~data_i[1])
226 if (~data_i[7])
227 state <= `DISABLE0;
228 else
229 state <= `STOP_ERR1;
230 else
231 state <= `WRITE11;
232 end
233
234 //ABILITO IL CORE
235 `DISABLE0: begin
236 add_reg <= `CTR;
237 data_o <= 8'b00000000;
238 w_rn <= 1'b1;
239 start_wb_cyc <= 1'b1;
```

```
240 state <= `DISABLE1;
241 end
242
243 `DISABLE1: begin
244 start_wb_cyc <= 1'b0;
245 if(got_ack)
246 state <= `IDLE0;
247 else
248 state <= `DISABLE1;
249 end
250
251 // In caso di mancanza di ACK dello slave mando lo STOP di chiusura
252 `STOP_ERR1: begin
253 state <= `STOP_ERR1;
254 end
255
256 `LOOP: begin
257 state <= `LOOP;
258 end
259
260 default:
261 state <= `INIT0;
262
263 endcase
264 end
265
266 ////////////////////////\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
267 // Macchina inferiore - interfaccia al wishbone bus \\
268 ////////////////////////\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
269 if (wb_rst_i) begin
270 wb_adr_o <= 3'h0;
271 wb_dat_o <= 8'h00;
272 wb_we_o <= 1'b0;
273 wb_stb_o <= 1'b0;
274 wb_cyc_o <= 1'b0;
275 got_ack <= 1'b0;
276 end
277 else begin
278 if(start_wb_cyc == 1'b1) begin
279 // Start Wishbone Cycle
280 wb_stb_o <= 1'b1;
281 wb_cyc_o <= 1'b1;
282 wb_adr_o <= add_reg;
283 wb_dat_o <= data_o;
284 wb_we_o <= w_rn;
285 got_ack <= 1'b0;
286 end
287 else if((wb_stb_o == 1'b1)&&(wb_ack_i == 1'b1)) begin
288 // End Wishbone cycle
289 wb_stb_o <= 1'b0;
290 wb_cyc_o <= 1'b0;
291 wb_we_o <= 1'b0;
292 got_ack <= 1'b1;
293 if(~wb_we_o) begin
294 data_i <= wb_dat_i;
295 end
296 end
297 end
298 end
299
300 endmodule
301
```

Il codice implementa una macchina a stati finiti che compie un ciclo di scrittura quando l'apposito segnale di trigger presenta un impulso.
Tale macchina a stati viene qui rappresentata in forma compatta, condensando gli stati simili per non appesantire troppo l'impatto grafico.
Ad esempio, i 13 stati per completare la scrittura (WRITE0-WRITE13) sono riuniti sotto un unico stato (WRITE).

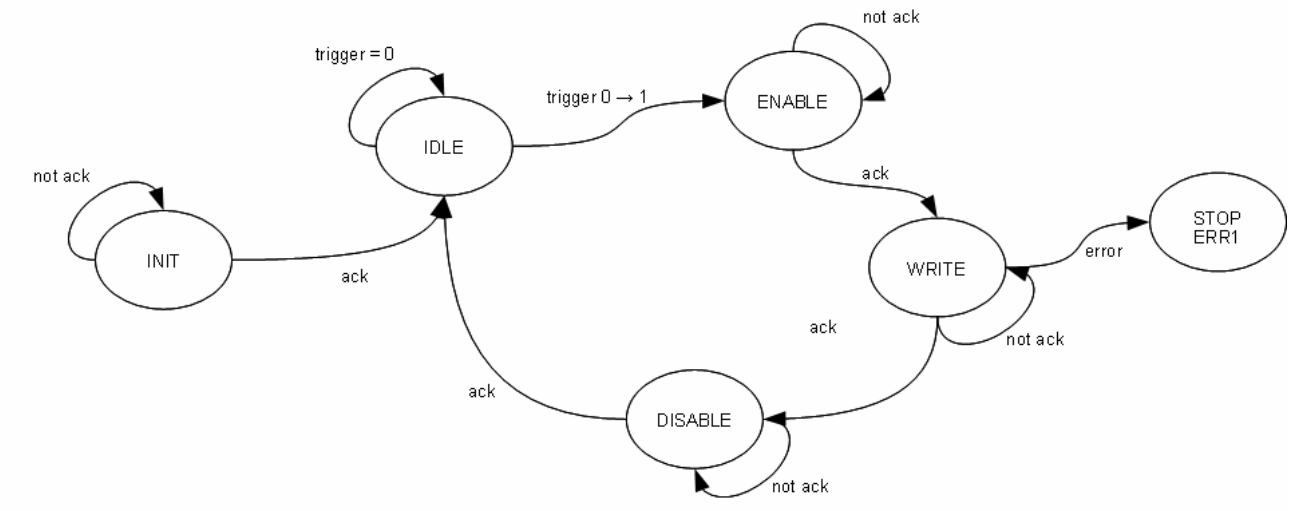
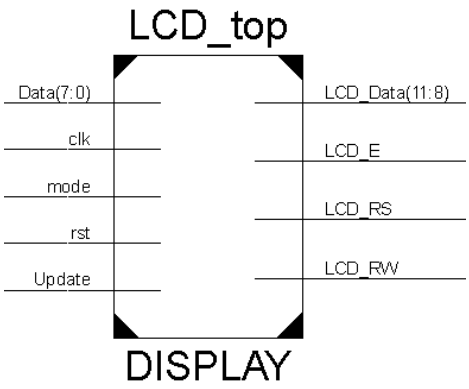


Fig. 3.15 – Macchina a stati finiti sintetica

Ciascuno dei blocchi indicati nel diagramma contiene al suo interno un certo numero di stati ma, sostanzialmente, il principio di avanzamento della macchina si basa ovunque sullo stesso algoritmo:

1. Scegliere l'**indirizzo** del registro con cui si vuole operare
2. Eseguire l'**operazione** sul Wishbone Slave, che può essere una scrittura o una lettura
3. Attendere l'**acknowledge**, ossia la conferma che l'operazione ha avuto successo, e in base a questo far progredire o meno lo stato della macchina

3.6 – Visualizzazione su display



Il blocco adibito alla scrittura sul display lavora in maniera parallela rispetto al Controller del bus I^2C.
Quindi, il sistema provvede a scrivere ogni nuovo valore di *AMP* contemporaneamente sia nel dispositivo HV892, sia nel display.
Per raggiungere questo risultato, si è scelto di aggiornare il display sul fronte di salita del segnale *Update*, che è lo stesso segnale impulsivo ad entrare nel modulo Wishbone Master con il nome *trigger*.
In definitiva, ogni volta che l'utente applica un input al sistema mediante gli switch a disposizione , il blocco *knob_interace* restituisce in uscita un segnale di trigger impulsivo e un valore aggiornato di *AMP*: tale coppia di segnali va in input sia al controller I^2C, sia al controller LCD.
Sul piano tecnico, anche il modulo *LCD_top* è stato implementato mediante macchine a stati finiti annidate, con livello di astrazione crescente con la gerarchia dei moduli.

Di seguito si riporta il codice del modulo top:

```
module LCD_top(
      input clk,
      input rst,
      input Update,
      input mode,
      input [7:0] Data,
      output [11:8] LCD_Data,
      output LCD_E,
      output LCD_RS,
      output LCD_RW
);

      wire ClkDiv;
      wire LCDAck;

      wire GenBusy;
      wire [8*16:1] DisplayData;

      reg LCDUpdate;
      reg [2:0] state;
      reg [7:0] AMP;
      reg Refresh;
      reg oldUpdate;

      LCDEncoder LCDStringGen(      .CLK          (clk               ),
                                    .RST          (rst               ),
                                    .Refresh      (Refresh           ),
                                    .Raw_Input    (AMP               ),
                                    .Busy         (GenBusy           ),
                                    .LCD_Output   (DisplayData       )
                                );

      clock_slower2 LCDClock  (     .clk          (clk           ),
                                    .reset        (rst           ),
                                     .slow_clk     (ClkDiv        )
                                );

     LCDDisplay LCDDriver     (     .Clk              (ClkDiv          ),
                                    .Rst              (rst             ),
                                    .LCDUpdate        (LCDUpdate       ),
                                    .Mode             (mode            ),
                                    .LCD_Data_In      (DisplayData     ),
                                    .LCDAck           (LCDAck          ),
                                    .LCD_Data         (LCD_Data        ),
                                    .LCD_E            (LCD_E           ),
                                    .LCD_RS           (LCD_RS          ),
                                    .LCD_RW           (LCD_RW          )
                                );


always @ (posedge clk or posedge rst)
if (rst)
      begin
      LCDUpdate <= 0;
      state <= 0;
      AMP <= 0;
      Refresh <= 0;
      oldUpdate <= 0;
      end
else
      case (state)

      0:
      begin
```

```
            oldUpdate <= Update;
            if (Update == 1 && oldUpdate == 0)
                  begin
                  state <= 1;
                  AMP[7:0] <= Data;
                  Refresh <= 1;
                  end
            else
                  state <= 0;
      end

      1:
      begin
            if (GenBusy)
                  state <= 1;
            else
                  state <= 2;
      end

      2:
      begin
            LCDUpdate <= 1;
            Refresh <= 0;
            if (LCDAck == 1)
                  begin
                  state <= 3;
                  LCDUpdate <= 0;
                  end
            else
                  state <= 2;
      end

      3:
      begin
            if (LCDAck == 0)
                  state <= 0;
            else
                  state <= 3;
      end

      default:
            state <= 0;
      endcase

endmodule
```

Nella testa del codice si possono individuare tre moduli che vengono istanziati: *LCDClock, LCDStringGen* ed *LCDDriver.*

Modulo LCDClock

Contiene un divisore di clock strutturalmente uguale a quello utilizzato per il sistema antirimbalzo.
La sua presenza è necessaria per il corretto funzionamento del display: infatti, nei capitoli precedenti si era fatto presente che l'LCD in dotazione alla Spartan fosse molto lento e che pertanto non sarebbe stato in grado di sostenere direttamente la velocità dell'oscillatore di sistema.
In fase di progettazione si è scelto di dividere il clock utilizzando un contatore a 11 bit: valori più elevati sarebbero stati un inutile dispendio di prestazioni, mentre per valori più bassi si era constatato un funzionamento scorretto del display, dato che la stessa inizializzazione non aveva successo.

Modulo LCDStringGen

Tale generatore di stringhe ha lo scopo di preparare il dato per essere inviato al display nel formato idoneo; le operazioni che svolge sono le seguenti:

1. Converte il valore binario a 7bit di *AMP* nelle rispettive cifre BCD che sono un formato compatibile con la mappa caratteri del controller LCD
2. Aggiunge la parte testuale al dato, distinguendo a seconda che il sistema sia in standby, resettato oppure in funzionamento normale
3. Porta il dato a 128bit (16 caratteri ciascuno di 8 bit) in uscita

Il codice che lo implementa è il seguente:

```
module LCDEncoder(
    input RST,
    input CLK,
    input Refresh,
    input [7:0] Raw_Input,
    output reg Busy,
    output reg [8*16:1] LCD_Output
    );

wire [11:0] BCD_Output;

//Scritta --> "NIX Standby"
parameter STANDBY = 128'B01001110...
parameter RESET_STRING = 128'B01001110...

Bin2BCD Converter (
                  .Binary     (Raw_Input),
                  .BCD  (BCD_Output)
             );

always @ (posedge CLK or posedge RST)
      if (RST)
            begin
                  Busy <= 1;
                  LCD_Output <= RESET_STRING;
            end
      else
            if (Refresh)
                  begin
                  Busy <= 1;
                  if (Raw_Input == 8'h00)
                        LCD_Output <= STANDBY;
                  else
                        begin
                              //Scritta --> "NIX OUT="
                              LCD_Output [128:65] <= 64'B01001110...

                              //Livello polarizzazione, es. "255"
                              LCD_Output [64:61] <= 4'B0011;
                              LCD_Output [60:57] <= BCD_Output[11:8];

                              LCD_Output [56:53] <= 4'B0011;
                              LCD_Output [52:49] <= BCD_Output[7:4];

                              LCD_Output [48:45] <= 4'B0011;
                              LCD_Output [44:41] <= BCD_Output[3:0];

                              //Spazi finali --> "    #"
                              LCD_Output [40:1] <= 40'B00100000...
```

```
                    end
                end
            else
                Busy <= 0;
endmodule
```

Modulo LCDDriver

Rappresenta il blocco logico che si interfaccia fisicamente con il display, ovvero che ha il controllo diretto dei relativi pin sull'FPGA.
Secondo le specifiche della Xilinx, il controller può interfacciarsi con il display utilizzando a scelta la modalità a 4 oppure ad 8 bit.
Dato l'utilizzo relativamente leggero che si fa del display, e visto che non ci sono particolari vincoli di velocità, tale interfaccia è stata implementata utilizzando il bus a 4 bit.
I restanti pin, corrispondenti ai bit meno significativi, sono stati collegati direttamente a livello logico alto in modo da predisporre questa modalità di funzionamento a bus ridotto.
L'architettura del modulo, trattandosi sempre di una scrittura, ricalca quella del modulo Wishbone Slave precedentemente analizzata, che ha lo scopo di scrivere i dati nel bus I^2C.
Analogamente a prima, infatti, sono individuabili all'interno del modulo diverse macchine a stati finiti che operano a livelli di dati diversi.
Tale modulo si presenta gerarchicamente così suddiviso:

- *LCDDriver* (LCDDisplay.v)
  - LCDInterface.v
    - LCDcntrl.v
    - LCDDriver.v

LCDDriver e LCDCntrl sono le entities che lavorano a basso livello, completando la scrittura nel seguente modo: LCDCntrl riceve la parola a 128 bit e in sequenza invia pacchetti da 8 bit (corrispondenti ad un singolo carattere nella mappatura del display) a LCDDriver, il quale si occupa di scriverli singolarmente sul display.
Si fa presente che il display, al momento dell'inizializzazione, è stato preconfigurato in modo da spostare automaticamente il cursore a destra di una posizione dopo ogni carattere scritto.

3.7 – Risorse utilizzate

L'ambiente di sviluppo Xilinx ISE ci consente, nella fase successiva alla progettazione, di avere un report stimato sull'utilizzo delle risorse fisiche, utile per avere un indice di quanto efficientemente è stato descritto l'hardware.
Nel nostro caso i riepiloghi di utilizzo in fase di sintesi (valori stimati) e implementazione (valori reali) sono stati i seguenti:

SINTESI

Device Utilization Summary (estimated values)			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slices	640	5888	10%
Number of Slice Flip Flops	780	11776	6%
Number of 4 input LUTs	1158	11776	9%
Number of bonded IOBs	20	372	5%
Number of GCLKs	2	24	8%

IMPLEMENTAZIONE

Device Utilization Summary				
Logic Utilization	**Used**	**Available**	**Utilization**	**Note**
Number of Slice Flip Flops	780	11,776	6%	
Number of 4 input LUTs	1,121	11,776	9%	
Number of occupied Slices	725	5,888	12%	
Number of Slices containing only related logic	725	725	100%	
Number of Slices containing unrelated logic	0	725	0%	
Total Number of 4 input LUTs	1,155	11,776	9%	
Number used as logic	1,121			
Number used as a route-thru	34			
Number of bonded IOBs	20	372	5%	
IOB Flip Flops	3			
Number of BUFGMUXs	2	24	8%	
Average Fanout of Non-Clock Nets	4.29			

Come si può osservare dai riepiloghi, l'utilizzo delle risorse a disposizione è nel complesso contenuto.
In ogni caso, trattandosi di un prototipo, e considerando che le versioni successive del prodotto non saranno implementate mediante FPGA, non ci sarebbe motivo per effettuare un'operazione di revisione del codice per ottimizzare lo sfruttamento delle risorse.

Capitolo 4
POLARIZZATORE

4.1 – Polarizzazione

In questo paragrafo sono trattate alcune informazioni di base riguardo la luce e la polarizzazione, per poter inquadrare meglio il campo di applicazione del progetto.
Per raggiungere questo scopo, prima della polarizzazione verranno dati alcuni richiami sulle onde elettromagnetiche.

4.1.1 – Classificazione delle onde

Il primo passo per avvicinarsi al polarizzatore ottico è quello di classificare i vari tipi di onda, evidenziando per ciascuno di essi se abbia senso o meno parlare di polarizzazione.
Una classificazione di base delle onde può essere effettuata osservando due caratteristiche fondamentali:

1. La *direzione* in cui si muovono le singole particelle del mezzo di propagazione
2. La capacità dell'onda di propagarsi nel *vuoto*

Una **prima distinzione** delle onde viene effettuata appunto a seconda della direzione in cui si muovono le particelle del mezzo in cui questa si propaga.
Prima di tutto è bene notare che con l'espressione “direzione di propagazione” ci si riferisce alla direzione nella quale avviene il trasporto di energia.
Le onde in generale possono essere *longitudinali* oppure *trasversali*: rientrano nel primo caso quei tipi di onda in cui le particelle del mezzo si muovono in direzione coincidente a quella di propagazione; le onde *trasversali* invece sono quelle in cui il tale movimento del mezzo avviene in direzione perpendicolare a quella di propagazione.

La **seconda distinzione** che si compie riguarda la capacità di un'onda di propagarsi nel vuoto: questa è una caratteristica tipica delle onde *elettromagnetiche*, vale a dire quelle onde prodotte da una carica elettrica in movimento.
È bene notare che le onde elettromagnetiche, dal punto di vista vettoriale, sono date dalla composizione di un campo elettrico e di un campo magnetico: questo è coerente, dal punto di vista fisico, con il fatto che derivino da cariche elettriche in moto.
Tutti i tipi di onde *meccaniche* non sono in grado di propagarsi nel vuoto ma richiedono un mezzo fisico attraverso il quale potersi spostare.

4.1.2 – Onde polarizzate

Se considero un fascio di luce proveniente da una sorgente qualsiasi, ad esempio una lampadina, con tutta probabilità esso sarà costituito da una moltitudine di singole onde, dove ciascuna oscilla su di un piano ben definito mantenendo comunque una direzione di propagazione comune.
Si tratta dunque di un fascio di onde *non polarizzato*.
Se invece prendo in esame una sola di queste onde, l'oscillazione avviene in un unico piano, e tale onda viene detta *polarizzata*.
In generale, la luce proveniente da sorgenti naturali è quasi sempre non polarizzata, visto che è causata dalla vibrazione di cariche elettriche in molte direzioni casuali.

Richiamando il paragrafo precedente, il **concetto di polarizzazione** ha senso solamente se si sta facendo riferimento ad onde *elettromagnetiche* e comunque *trasversali*.
Con il termine *polarizzazione* si va ad indicare il processo di trasformazione di un generico fascio luminoso in un'onda polarizzata, operazione che si può effettuare in diversi modi, ad esempio:

1. Polarizzazione per trasmissione
2. Polarizzazione per riflessione
3. Polarizzazione per rifrazione

TRASMISSIONE

La tecnica più comune per polarizzare un fascio di luce è quello di farlo passare attraverso un filtro, come ad esempio il Polaroid.
Il materiale di questi filtri è in grado di far passare solamente le onde che oscillano su di un piano ben definito, assorbendo con buona efficacia tutte le altre.
Questa caratteristica è riconducibile ad una proprietà chimica di tale materiale: esso infatti è costituito da lunghe catene di molecole che vengono stirate per tutta l'estensione del filtro, e disposte in maniera omogenea sulla sua superficie.
Come test per il funzionamento, è possibile sovrapporre due filtri Polaroid in modo che i loro assi di polarizzazione siano perpendicolari: il risultato è che il primo filtro restituisce un'onda che oscilla su un piano, e il secondo filtro, avendo asse di polarizzazione ortogonale, la assorbe completamente; pertanto l'azione combinata di questi due filtri è quella di oscurare la radiazione iniziale.
Il Polaroid rappresenta l'esempio più comune di un filtro polarizzatore in trasmissione di tipo passivo; nel caso del progetto del polarizzatore elettronico, si è scelto di utilizzare un filtro trasmissivo di tipo attivo, vale a dire una cella LCD.
Come verrà spiegato più avanti, anche i cristalli liquidi possono ricoprire la stessa funzione del Polaroid, con la differenza che l'asse di polarizzazione può essere variato per via elettronica, senza dover necessariamente agire meccanicamente sul filtro.

RIFLESSIONE E RIFRAZIONE

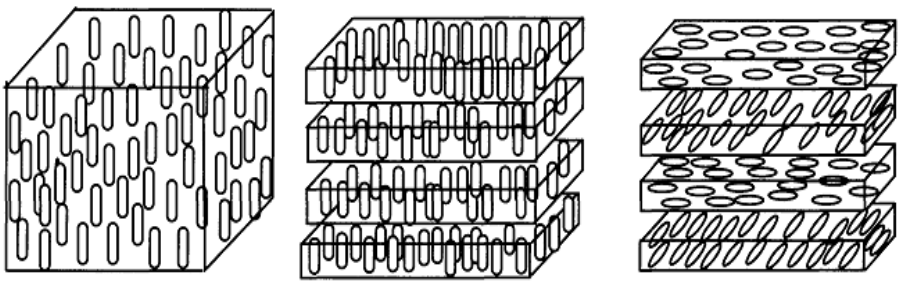
Un altro modo per ottenere luce polarizzata a partire da un generico fascio luminoso è quello di sottoporla a riflessione da parte di una superficie non metallica.
La radiazione riflessa risulterà polarizzata parallelamente alla superficie: è bene notare che l'efficacia della trasformazione dipende principalmente dall'angolo d'incidenza del fascio originale e dal materiale della superficie riflettente.
Come specificato in precedenza, le superfici metalliche non si prestano per funzionare da polarizzatori in riflessione: questo si spiega considerando che i metalli, una volta raggiunti da una radiazione elettromagnetica, sono in grado di rifletterla in una varietà di direzioni differenti.
Un buon esempio di luce polarizzata riflessa è quella proveniente dalle superfici d'acqua: in fotografia, infatti, è cosa frequente impiegare dei polarizzatori ottici proprio allo scopo di eliminare i riflessi indesiderati provenienti dal mare o altri specchi d'acqua.
Un fenomeno simile si verifica quando il fascio non polarizzato viene rifratto nel passaggio tra due materiali aventi densità differente: in questo caso tuttavia la radiazione riflessa che si ottiene è polarizzata perpendicolarmente alla superficie di passaggio.
È da osservare che i due fenomeni appena descritti (polarizzazione per riflessione e rifrazione) sono stati riportati solo allo scopo di accennare quali possono essere le fonti naturali di luce polarizzata: dovendo progettare un filtro, le soluzioni in trasmissione sono senza dubbio preferibili sia per la loro comodità che per la maggiore efficacia polarizzante.
La luce riflessa oppure rifratta infatti presenta solitamente un certo grado di polarizzazione, che tuttavia non è elevato come nel caso di un filtro trasmissivo appositamente studiato.

4.2 – Cristalli liquidi

Nel progetto del polarizzatore elettronico viene impiegato un filtro trasmissivo di tipo attivo, che si basa sulla tecnologia LCD.
Con il termine *cristalli liquidi* si indicano dei *composti organici* dotati di proprietà fisiche particolari; la definizione stessa suggerisce che essi presentino caratteristiche appartenenti ad ambedue gli stati solido e liquido, al punto che viene loro attribuito uno stadio intermedio tra questi, detto *mesofase*.
Questi materiali sono formati da molecole ellissoidali la cui disposizione, pur essendo vincolata ad un certo ordine geometrico (come avviene nei solidi cristallini), permette comunque un certo grado di libertà spaziale delle molecole: ad esempio esse sono in grado di ruotare e traslare l'una rispetto all'altra, in seguito a sollecitazioni meccaniche o elettriche.
Le proprietà che tornano più utili nelle applicazioni ottiche sono senza dubbio l'anisotropia elettrica e la possibilità di regolarne l'ordine mediante l'applicazione di campi magnetici ed elettrici.

I cristalli liquidi sono classificabili in tre mesofasi:

1. **NEMATICA**
   le molecole si presentano tutte parallele tra loro, ma posate su piani casuali
   (3 gradi di libertà : traslazione in x,y,z)
2. **SMECTICA**
   le molecole sono ancora parallele tra loro, ma posano su piani ben definiti
   (2 gradi di libertà per ogni piano : traslazione in x,z)
3. **COLESTEROLICA**
   le molecole sono parallele tra loro a piani alterni
   (2 gradi di liberta per ogni piano : traslazione in x,z)



(1) NEMATICA (2) SMECTICA (3) COLESTEROLICA

Fig. 4.1 – Mesofasi più comuni dei cristalli liquidi

Azione del campo elettrico
Le molecole elissoidali dei cristalli liquidi reagiscono all'applicazione di un campo elettrico trasformandosi in dipoli indotti con le cariche concentrate alle estremità.
Il campo elettrico agisce dunque sulle molecole come una coppia di forze, e tende ad orientarle con l'asse parallelo alle linee di forza.

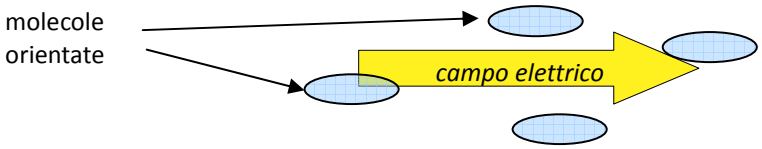


Fig. 4.2 – Azione del campo elettrico sulle molecole dei cristalli liquidi

4.3 – Cella LCD

La cella impiegata nel progetto contiene cristalli liquidi di tipo *nematico*, dunque tutte le molecole che lo compongono sono disposte casualmente nello spazio, con l'unico vincolo di essere, a riposo, parallele tra loro secondo l'asse longitudinale.
La sezione di tale cella consente di capire meglio quanti strati la compongono e quale funzione essi svolgano.

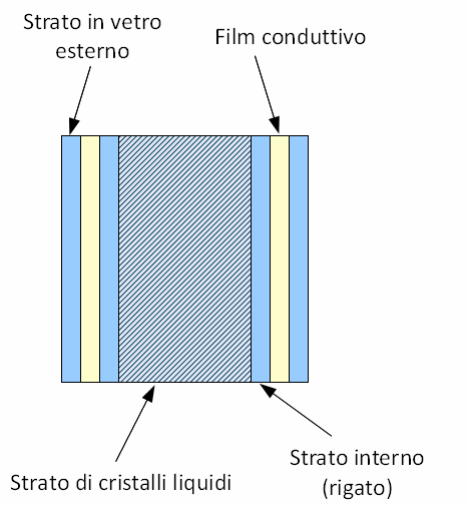


Fig. 4.3 – Sezione di una cella a cristalli liquidi

Tale cella è così costruita:

- interno riempito di cristalli liquidi
- 2 pareti esterne in vetro
- 2 strati interni di allineamento
- 2 strati intermedi conduttivi e trasparenti

Le due pareti poste all'esterno del dispositivo hanno il solo scopo di contenere fisicamente tutti gli altri componenti.
Appena più internamente, vengono collocati due sottilissimi film caratterizzati da elevata conducibilità elettrica: così facendo, una volta collegati ai capi di un generatore di tensione, si induce un campo elettrico uniforme su tutta la superficie della cella.
Nella parte interna della cella viene posto uno strato di cristalli liquidi in forma nematica, che tuttavia non va a contatto direttamente con le lamine conduttive: infatti, viene interposto un ulteriore strato che viene opportunamente abraso.
Tale abrasione viene praticata in modo molto leggero, e avviene ovunque nella stessa direzione: la

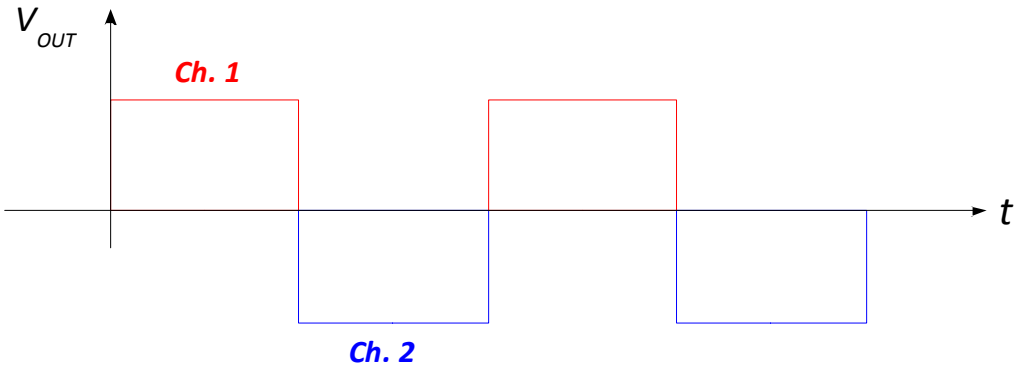
finalità di questa operazione è quella di conferire un orientamento preferenziale alle molecole dei cristalli liquidi.
Infatti, essendo questi a diretto contatto con la parete, succede che tendono ad adagiarsi su di essa seguendo proprio l'orientamento dettato da questi graffi: in assenza di campo elettrico, quindi, si può pensare che tutte le molecole dei cristalli liquidi abbiano una direzione ordinata.
Bisogna precisare che non solo le molecole a contatto con la superficie graffiata si orientano, bensì l'intera massa dei cristalli liquidi: trattandosi infatti di una varietà nematica, le molecole tenderanno ad assumere un orientamento omogeneo anche nella parte più interna dello strato, e a restare sempre parallele tra loro.

Capitolo 5
TEST

Nel seguente capitolo vengono riportate le osservazioni conseguenti i vari test a cui è stato sottoposto il sistema nella fase finale.

TEST ELETTRONICI

Una prima verifica del corretto funzionamento è stata quella di controllare se l'onda quadra generata all'uscita dell'elettronica di controllo rispettasse tutti i parametri indicati nelle specifiche.
A tale scopo si è utilizzato un oscilloscopio digitale collegato ai morsetti OUT1 e OUT2 dell'integrato HV892.
Dal momento che l'uscita OUT1 è quella che produce la semionda quadra positiva e OUT2 quella negativa, sono state connesse ciascuna ad un canale dell'oscilloscopio e successivamente, mediante le funzioni matematiche messe a disposizione dello strumento, si è generato un segnale virtuale dato dalla somma dei due canali, che corrisponde all'onda quadra cercata.
Tutte le misurazioni successive sono state effettuate considerando direttamente questo segnale virtuale.



Nella seguente tabella riassuntiva vengono riportati i parametri che sono risultati più significativi per valutare l'accuratezza dell'onda quadra prodotta.
Alcune di queste grandezze, come evidenziato, presentano un certo scostamento dal valore nominale riportato nelle specifiche: questo problema è da ricondursi al fatto che, con l'oscilloscopio utilizzato, le sonde corredate presentavano un'impedenza di ingresso di 1MΩ, a differenza dei 10MΩ che venivano suggeriti nel datasheet della Supertex per le eventuali misurazioni.

Parametro	Valore rilevato	Range datasheet
Frequenza	≈ 1.42 kHz	1.0 ÷ 2.0 kHz
Ampiezza AMP=00xH	≈ 0.15 V_{RMS}	0 V_{RMS}
Ampiezza AMP=FFxH	≈ 56 V_{RMS}	58.5 ÷ 65.5 V_{RMS}

TEST OTTICI

I test su banco ottico non sono stati al momento effettuati a causa dell'impossibilità di reperire in tempo utile la cella a cristalli liquidi compatibile con il sistema.

CONCLUSIONI

Al termine del tirocinio, gli obiettivi prefissi sono stati interamente rispettati: la realizzazione del prototipo è stata completata sviluppando tutte le funzionalità pensate originariamente, ed inoltre è stata aggiunta una modalità, utile in fase di debug, con la quale è possibile aumentare lo step di incremento/decremento dell'ampiezza nell'onda quadra in uscita.
Questa funzionalità risulta utile per stressare maggiormente la componente dinamica del sistema.
Dal punto di vista personale, ritengo che il tirocinio si sia svolto in maniera soddisfacente, principalmente per due motivi: anzitutto, dal punto di vista tecnico, avendo avuto l'opportunità di entrare in contatto con la parte più concreta e commerciale dell'elettronica; infatti mi sono fatto un'idea più chiara di come vengano costruiti i sistemi elettronici, servendosi di componenti di vari produttori opportunamente interfacciati tra loro.
A questo proposito, è stato fondamentale investire del tempo per reperire ed analizzare accuratamente i data-sheet degli integrati e le documentazioni dei bus standard (I^2C, Wishbone..).
Questa esperienza assume ancora più valore poiché, avendo frequentato una scuola superiore ad orientamento umanistico, e del tutto priva della componente tecnico-pratica, ho potuto acquisire familiarità con alcuni aspetti di base dell'elettronica che risultano invece banali o scontati per chi proviene da un istituto superiore di tipo tecnico.
In secondo luogo, l'altro vantaggio derivante dalla frequentazione del tirocinio, e che ritengo non meno importante dei progressi didattici raggiunti, è stato quello di poter osservare e toccare in prima persona lo svolgimento dell'attività aziendale: trovandomi sempre in contatto con il personale, infatti, ho avuto modo di familiarizzare con le dinamiche e i problemi che si affrontano quotidianamente nell'attività produttiva.
Oltretutto, il tirocinio si è svolto in un'azienda fondata da poco, il che è stato senza dubbio un vantaggio, poiché, parallelamente alla realizzazione del progetto assegnatomi, avevo la possibilità di seguire da vicino la produzione principale, che è al momento particolarmente intensa dato che l'impresa è in fase di sviluppo.

Fonti

- **Manuale utente Spartan 3A Kit**
  (http://www.xilinx.com/support/documentation/boards_and_kits/ug334.pdf)

- **Specifiche OpenCores I^{2}C Controller Core**
  (http://www.opencores.org/project,i2c)

- **Wishbone Specification edizione Settembre 2002**
  (http://www.opencores.org/downloads/wbspec_b3.pdf)

- **Specifiche Bus I^{2}C Philips edizione 2.1 Gennaio 2000**
  (http://www.nxp.com/acrobat_download2/literature/9398/39340011.pdf)

- **Optics**
  E. Hecht, A. Zajac
  Addison Wesley